mi COMPUTER

33

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR



Editorial Delta, S.A.

150 ptas.

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona, y comercializado en exclusiva por Distribuidora Olimpia, S.A., Barcelona

Volumen III - Fascículo 33

Director:	José Mas Godayol
Director editorial:	Gerardo Romero
Jefe de redacción:	Pablo Parra
Coordinación editorial:	Jaime Mardones Francisco Martín
Asesor técnico:	Jesús Nebra

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti. A. Cuevas, F. Blasco

Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, Barcelona-8
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador,* se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-85822-94-3 (tomo 3)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 298408
Impreso en España - Printed in Spain - Agosto 1984

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, Madrid-34.
Distribuye para Argentina: Viscontea Distribuidora, S.C.A., La Rioja 1134/56, Buenos Aires.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio Blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Ferrenquín a Cruz de Candelaria, 178, Caracas, y todas sus sucursales en el interior del país.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 16 690 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 087 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 3371872 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Distribuidora Olimpia (Paseo de Gracia, 88, 5.°, Barcelona-8), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Distribuidora Olimpia, en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.





La ley del más apto

Millones de microprocesadores controlan las más diversas aplicaciones, pero sólo dos diseños dominan el mercado: el Z80 y el 6502

Los ordenadores de un chip fueron casi un resultado accidental. En 1972 Datapoint pidió a Intel —la empresa fabricante de chips— que ideara uno capaz de reemplazar los numerosos chips TTL (transistor-transistor-lógico) que empleaban los terminales de ordenador de la época. La pieza que produjeron se llamó 8008. Podía procesar datos de a ocho bits por vez y habría sido el "reemplazo lógico" ideal para los terminales de Datapoint de no ser por un inconveniente: operaba con demasiada lentitud. A pesar de que Datapoint decidió no adoptarlo, ingenieros y aficionados pronto repararon en la capacidad potencial del 8008 como la CPU para un ordenador de uso general y así nació el ordenador para una mesa de oficina.

Como pronto se reparó en las limitaciones del 8008 en lo referente a velocidad y potencia, Intel se propuso diseñar un sustituto. El chip que desarrollaron, el 8080, se convirtió rápidamente en el rey del mercado.

Aproximadamente en la misma época en que Intel dio a conocer el 8080, sus competidores de Motorola lanzaron al mercado un microprocesador de ocho bits conocido como el 6800. Si bien los conceptos de diseño del 8080 y del 6800 eran bastante distintos, ambos microprocesadores resultaban igualmente potentes y adecuados a su función de servir de base al diseño de un microordenador.

Aunque el 8080 y el 6800 eran igualmente eficaces, un accidente histórico allanó el terreno para el fenomenal éxito de un tercer chip, el Z80. En 1974, Gary Kildall —actualmente presidente de Digital Research— produjo para Intel un sistema operativo de disco denominado CP/M. Ello permitió que los ordenadores basados en el 8080 pudieran utilizar las recién introducidas unidades de disco flexible Shugart. Intel rechazó el sistema operativo de Kildall porque consideró que el software existente bastaba para ser utilizado con los sistemas de ordenador de unidad principal estándar de la época.

No obstante, los ordenadores pequeños se volvían cada vez más populares y el CP/M facilitó en gran medida el tratamiento de archivos de dichos sistemas. Este hecho aseguró durante años el dominio del mercado por el 8080 y dejó relativamente al margen al Motorola 6800. Se hicieron varios intentos de crear sistemas operativos de disco parecidos para el 6800, pero todo el impulso se volcó en el 8080 y el 6800 quedó en segundo plano.

A medida que crecía el mercado de productos basados en el microprocesador, los fabricantes de chips luchaban por ofrecer nuevos diseños, pero siempre se encontraban con una barrera: la reticencia del mercado para aceptar lo nuevo a menos que supusiera ventajas considerables. Las inversiones en diseño de hardware y en producción de software también frenaron la adopción de cualquier nuevo microprocesador incompatible.



La elección vital
La mayoría de los micros personales eligen entre el procesador 6502 (como el BBC Micro) o el Z80 (p. ej., el Spectrum). El Dragon, una de las pocas máquinas que utiliza otros chips, incorpora el 6809

Una idea genial dio la oportunidad inesperada a un nuevo diseño de chip: el Z80. Zilog —un equipo de ingenieros de diseño que anteriormente habían trabajado para Intel en el 8080— se dio cuenta de que era posible ampliar el conjunto de instrucciones. En síntesis, no se habían aprovechado todas las combinaciones posibles de unos y ceros que el 8080 podía reconocer como instrucciones. Mediante el empleo de combinaciones binarias no utilizadas por el chip de Intel, Zilog logró diseñar un microprocesador capaz de funcionar igual que el 8080 cuando se le proporcionaban instrucciones específicas de éste pero que además podía ofrecer una considerable mejora en rendimiento. De este modo consiguieron crear un chip que utilizaba software escrito para el 8080.

Además de esta innovación, Zilog presentó otra importante ventaja comercial. Mientras que el chip de Intel dependía de un chip de generador de reloj

Chips en croquis

Los microprocesadores evolucionaron a partir de dos fuentes principales: los que surgieron de los microprocesadores originales de Intel y los que se desarrollaron a partir del chip 6800, de la empresa rival Motorola. Este esquema muestra el modo en que se desarrollaron los chips, así como algunas de las máquinas a las que fueron incorporados. Muchos de los chips poco conocidos aparecen en los micros menos populares. Es posible que el Apple III sea la única máquina de oficina que utiliza un procesador 6502. El Olivetti M20 es el único micro de uso general que utiliza un Z8000. En ambos casos, la elección de un microprocesador poco común y su posterior carencia de software han inhibido el éxito de la máquina. Algunos ordenadores de enorme éxito, como el IBM PC, logran popularizar por sí solos un chip

'71 '72 '73 '74 '75 '76 '77 '78 '79 '80 '81 '82 '83 '84

Intel

4004

8008

4004 y 8008: Diseñado para reemplazar grandes cantidades de circuitos integrados de TTL, el 4004 era un chip muy sencillo que sólo podía manipular datos en grupos de cuatro bits. Intel abordó rápidamente el procesamiento con ocho bits mediante el 8008. Aficionados e ingenieros reconocieron su potencial y comenzaron a crear sus propios ordenadores de "fabricación casera"

Motorola

6800

6800: Es el rival del 8080. Con capacidades semejantes pero una concepción de diseño totalmente distinta. Aparecieron dos escuelas: la de los que prefirieron el enfoque del 8080 de Intel y la de los que escogieron el modo de trabajar del 6800 de Motorola

6809

6809: El perfeccionamiento que Motorola hizo del 6800 se tradujo en el 6809, probablemente el más capaz de todos los chips de ocho bits. Sin embargo, apareció demasiado tarde para causar impacto y sólo ha sido utilizado en unas pocas máquinas


Dragon 32

MOS Technology

6502

Commodore PET

6502: MOS Technology diseñó su propio chip de ocho bits que, pese a no ser compatible con el 6800, derivaba en gran parte de éste. Su bajo precio lo volvió muy atractivo para aficionados y diseñadores y por ello fue utilizado en la primera generación de máquinas tan vendidas como PET y Apple. Sigue siendo una elección popular para micros personales y ha sido empleado en ordenadores como el Oric y el BBC

Apple III

8080

8085

8080: Con la llegada del sistema operativo CP/M, éste se convirtió en el primer chip que podía emplearse para construir micros personales y de oficina. Al actualizar Intel el 8080, nació el 8085, compatible con el primero, con funciones adicionales y menos chips de soporte. El 8085 es asequible en una versión CMOS de baja potencia a menudo presente en máquinas portátiles como el Tandy Modelo 100


Tandy TRS-80 Modelo 100

Zilog

Z80

Z80: Un equipo de diseñadores abandonó Intel y formó Zilog con el propósito de producir esta versión perfeccionada y totalmente compatible del 8080


Tandy TRS-80 Modelo 1

68000

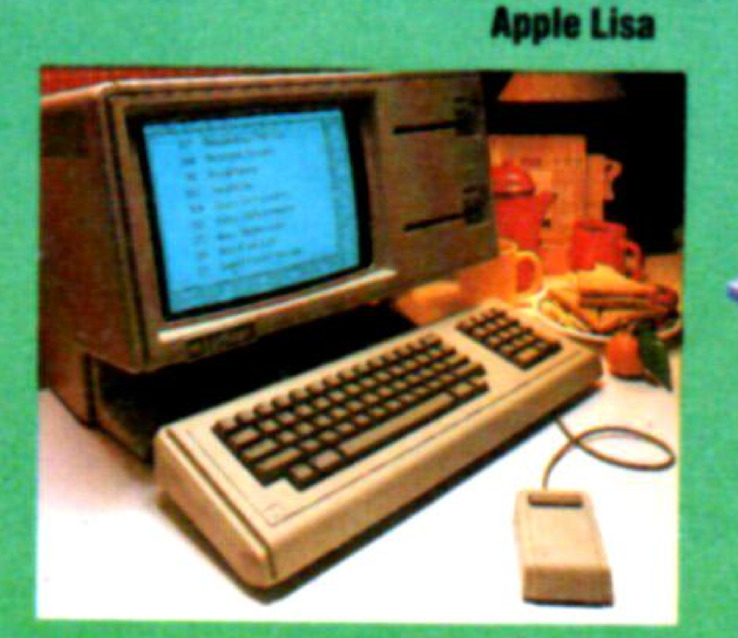
68000: El éxito del tan aclamado chip de 16 bits de Motorola se ha visto obstaculizado por la falta de software barato y el dominio del 8088. No obstante, Sinclair ha elegido la versión reducida del 68008 para su QL


68008

Sinclair QL



Apple Lisa



IBM PC

8086

8088

8088 y 8086: El 8088 es una versión reducida que puede emplear chips de soporte más antiguos, razón por la que durante un tiempo fue el chip de 16 bits más conocido. Su utilización en el Sirius y en el IBM PC lo convirtió en el más popular de los chips de 16 bits. La mayoría de las máquinas utilizan ahora el 8086, de mayor rendimiento, pero plenamente compatible

Z8000

Z8000: El primer chip de 16 bits de Zilog no ha sido popular en el mercado de ordenadores de uso general. Tal vez ello responda a un mal momento de salida y a la adopción del 8088 por IBM. El Z800 es el segundo intento de Zilog de producir un chip de 16 bits, que promete ser del todo compatible con el Z80 (y, por ende, con el 8080)


Olivetti M20



específico así como de un chip de control del sistema, el equipo Zilog logró combinar en un solo chip toda la lógica necesaria para un ordenador basado en un microprocesador. A pesar de que resultaba relativamente caro, el hecho de que pudiera reemplazar varios chips que cumplían otras funciones lo volvió muy atractivo para los fabricantes.

Aunque el 6800 había tenido poca suerte en comparación con el 8080, no por ello dejaba de ser popular entre algunos diseñadores y programadores. Más adelante Motorola diseñó un microprocesador de ocho bits altamente sofisticado, conocido como el 6809, que superaba al 6800. Lamentablemente, cuando el 6809 salió al mercado, una empresa rival llamada MOS Technology ya había presentado un nuevo perfeccionamiento del 6800 denominado 6502. Éste es el más popular de todos los procesadores incluidos dentro de lo que se conoce como la serie 6500. En dicha serie, la totalidad de sus integrantes utiliza el mismo conjunto de instrucciones, pero se diferencian unos de otros en cuanto a potencia y capacidades.

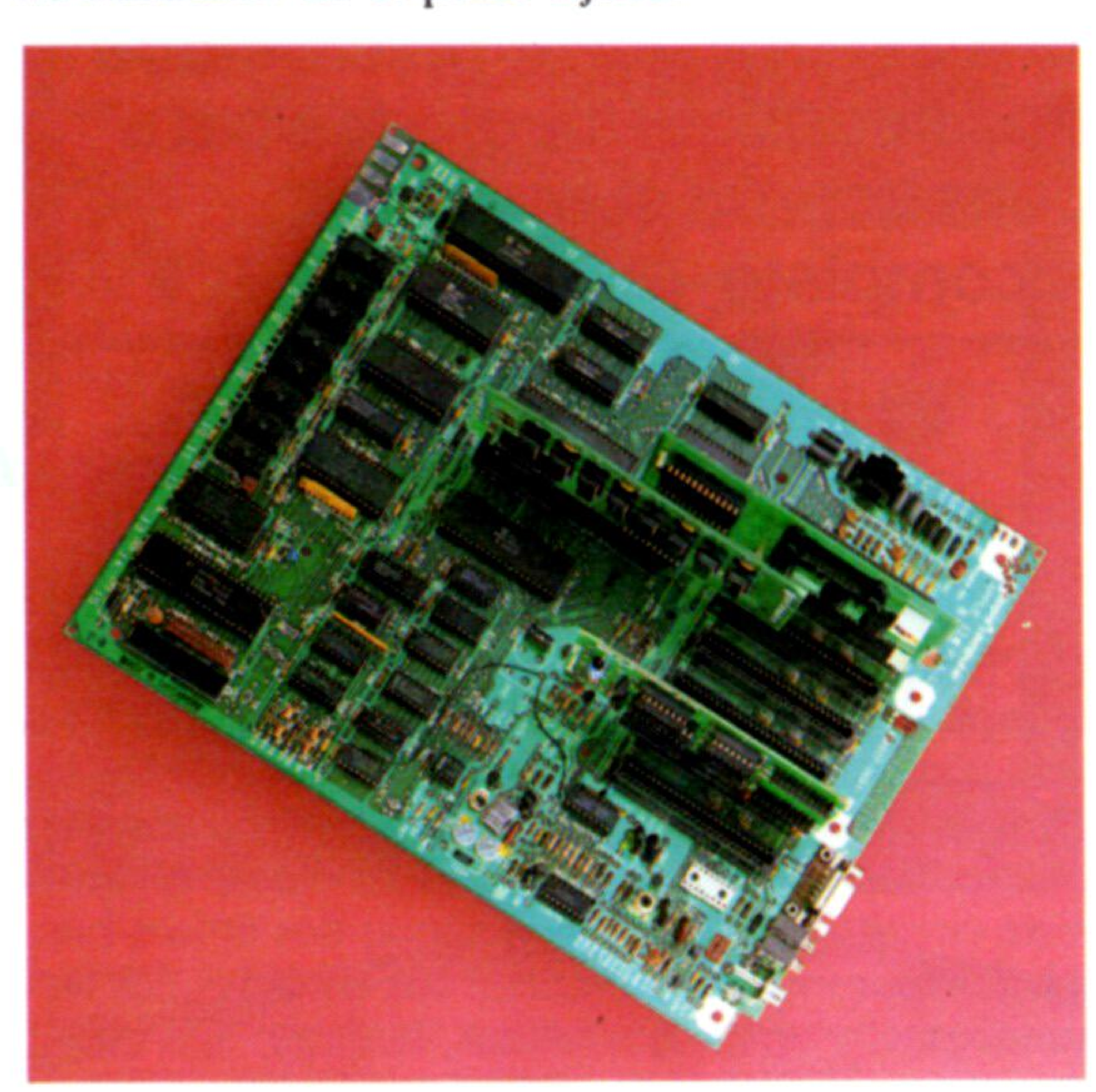
El 6502 de MOS Technology sigue un criterio de diseño de espíritu muy próximo al del 6800 de Motorola, pero no es compatible con éste ni en necesidades de hardware ni en compatibilidad de software. Por su parte, el Z80 incorpora todo el conjunto de instrucciones del 8080 y puede reemplazarlo en un sistema informático, si bien en este caso hay que realizar profundas modificaciones de diseño.

El 6502 ofrece un conjunto de instrucciones con el que cualquier programador del 6800 se sentiría a sus anchas, así como capacidades avanzadas y necesidades de interface ligeramente más sencillas. Sin embargo, no ofrece compatibilidad de software ni la posibilidad de un reemplazo chip por chip. Dados estos hechos, resultaría difícil imaginar que el 6502 gozara de su destacada posición actual de no ser por otro golpe de suerte: el 6502 fue utilizado en el ordenador Apple, que ha tenido un éxito sin precedentes.

Cuando apareció el Apple, los microordenadores de mesa estaban dominados por los diseños de bus basados en el S-100. Dichos micros se basaban en un circuito principal que transmitía potencia y señales a un circuito apropiado para cada función a cumplir. En consecuencia, un sistema S-100 mínimo requeriría una fuente de alimentación, un circuito principal, un circuito de CPU, otro de memoria, un tercero de VDU y probablemente un circuito para impresora y otro separado para unidades de disco. Es fácil ver lo costoso que sería un sistema S-100 en comparación con un sistema de un solo circuito, como el Apple.

Aunque el ordenador era relativamente barato, el principal adelanto de Steve Wozniak y su equipo de Apple tuvo lugar con un elemento de software de aplicaciones llamado VisiCalc. Este programa alcanzó gran popularidad entre los hombres de negocios, pues se dieron cuenta de que podían utilizarlo para generar presupuestos financieros con más rapidez y facilidad que si acudieran a la calculadora, lápiz y papel. VisiCalc tuvo tanto éxito que dio pie a que Apple vendiera masivamente su ordenador y de este modo el 6502 se estableció como uno de los principales diseños de microprocesador. Commodore también optó por el 6502 para el PET y sus sucesores.

Todavía se dio en Gran Bretaña un nuevo paso adelante cuando Acorn produjo el BBC Micro, que también se basa en este chip. Originalmente se había determinado que incorporara un Z80, pero ningún fabricante británico logró presentar un diseño adecuado en el plazo fijado.



Los chips cuentan
Los chips sofisticados reducen la cantidad de chips necesarios en un circuito. Cuando Apple mejoró el Apple II, la nueva versión IIe contaba con la mitad de chips principales

Mientras el chip 6502 establecía su dominio en el diseño de ordenadores de ocho bits, en el mercado comenzaron a aparecer ordenadores de 16 bits. Intel ofreció para estos ordenadores el 8088 y el 8086, al tiempo que Motorola producía el 68000 y Zilog el Z8000. Aunque estos tres diseños de 16 bits tienen sus méritos, ninguno es compatible con sus antecesores de ocho bits. Afortunadamente para Intel, Digital Research y Microsoft se apresuraron a ofrecer sistemas operativos para el 8086/8088 (CP/M-86 y MS-DOS, respectivamente), al tiempo que Zilog y Motorola fueron muy mal atendidos por el ramo del software. El hecho de que IBM adoptara el 8088 en su ordenador PC también ha dado un nuevo estímulo al chip de Intel.

La lucha por el dominio del mercado entre los chips de 16 bits promete ser una repetición de la historia del chip de ocho bits. Al igual que el Z80 y el 6502, el 8086 de Intel (y su versión reducida, el 8088) se han estandarizado. Entre las principales razones que dan cuenta de este hecho figuran el soporte de software de los sistemas operativos MS-DOS y CP/M-86 y su elección por los micros más vendidos, sobre todo el IBM y el Sirius. El chip Z8000 de Zilog sólo ha sido utilizado en un micro de uso general: el Olivetti M20. La Olivetti luchó por proporcionar software a la máquina y finalmente lanzó una ficha con un 8086 para que pudiera utilizar software MS-DOS y CP/M-86. Desde entonces, Zilog se ha propuesto diseñar un nuevo chip, el Z800, que no sólo cuenta con 16 bits, sino que puede utilizar software basado en el procesador Z80.

A pesar del creciente y rápido desarrollo en el campo de los chips de 16 bits, la mayoría de los ordenadores que se venden actualmente se basan en los diseños de ocho bits del Z80 y del 6502. Indudablemente, los ordenadores de 16 bits ofrecen ventajas de velocidad y potencia con respecto a sus antecesores, si bien en vista de la ingente cantidad de software que ya se ha desarrollado, las máquinas de ocho bits aún cuentan con una vida prolongada.

Medidas a medias

En esta ocasión emplearemos dos circuitos integrados para construir un sumador incompleto

CIRCUITO LÓGICO

A B OR AND NOT AND S C

Liz Dixon

Las puertas lógicas individuales que realizamos en el capítulo anterior son la base de circuitos digitales más complejos. Uno de esos grupos de puertas lógicas es el sumador incompleto, que analizamos en una lección de *Ciencia informática* (véase p. 513). Este circuito se emplea para sumar dos únicos bits. El sumador incompleto utiliza dos entradas, los bits sumandos, y ofrece dos salidas, la suma, más un bit de arrastre. La tabla de verdad que lo representa es la siguiente:

A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

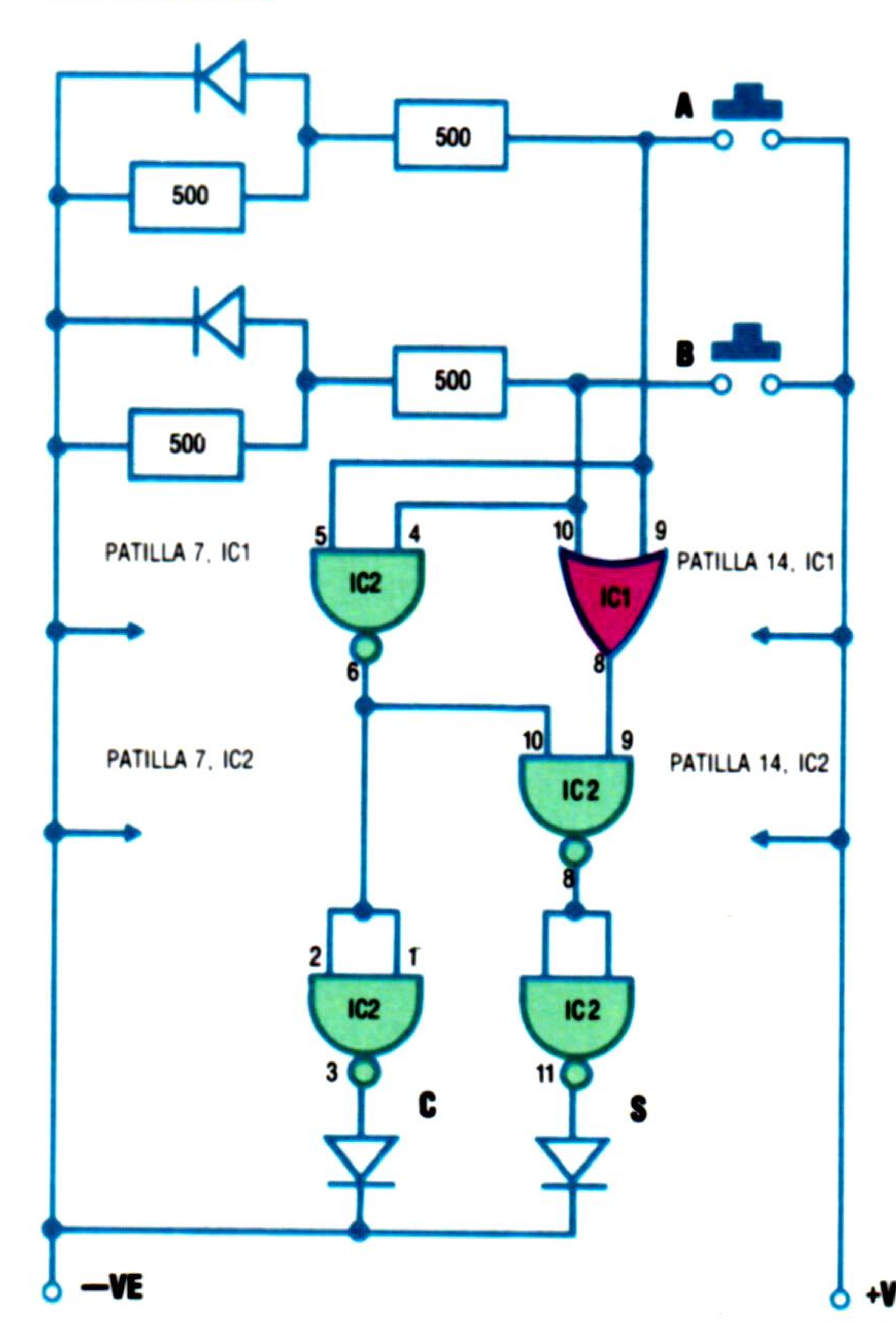
La salida S es la suma de los dos bits de entrada. Cuando ambos bits valen uno, la suma en binario es 10. Este resultado no puede representarse con un único bit de salida, ya que ésta se desborda en un segundo bit. Dicho desbordamiento es el bit de arrastre.

El sumador incompleto no resulta de mucha utilidad en los ordenadores de ocho bits; en realidad, lo que hace falta es un circuito que sume dos palabras de ocho bits. Dicho circuito puede construirse tomando como base 16 sumadores incompletos. Los dos primeros bits se suman mediante el primer sumador incompleto y el bit de la suma forma el primer bit del resultado. El bit de arrastre se añade al resultado de la segunda suma, el arrastre de dicha suma al tercero y así sucesivamente, enlazándolos de este modo.

Un sumador incompleto necesita alrededor de 10 transistores en las puertas que ya hemos construido. Sin embargo, las puertas lógicas AND, NAND, OR, NOR y otras son asequibles a muy bajo costo en grupos de cuatro en circuitos integrados simples. A partir de ellos, este sumador puede construirse más simplemente.

El circuito lógico de un sumador es el que mostramos a continuación. Se trata de la forma más sencilla de circuito. Utiliza tres tipos de puertas lógicas: OR, AND y NOT. Puesto que cada uno de los circuitos integrados que emplearemos sólo contiene un único tipo de puerta, el circuito lógico ha sido simplificado para utilizar menos puertas distintas. El circuito que construiremos utiliza cuatro puertas NAND y una sola puerta OR. El número de circuitos integrados se ha reducido a dos. Este circuito es más complejo que el circuito de una sola puerta que construimos en la página 624; será preciso, pues, prestar especial atención para cerciorarse de que todos los componentes quedan correctamente situados en el tablero de montaje de prueba.

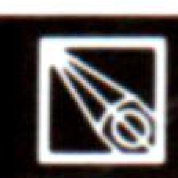
En cuanto haya terminado de construir este circuito, es posible que piense que es demasiado el esfuerzo para resultados tan simples. Aunque es mucho más fácil que construir el circuito a partir de componentes separados como son los transistores,



resulta difícil imaginar un ordenador entero construido de esta forma.

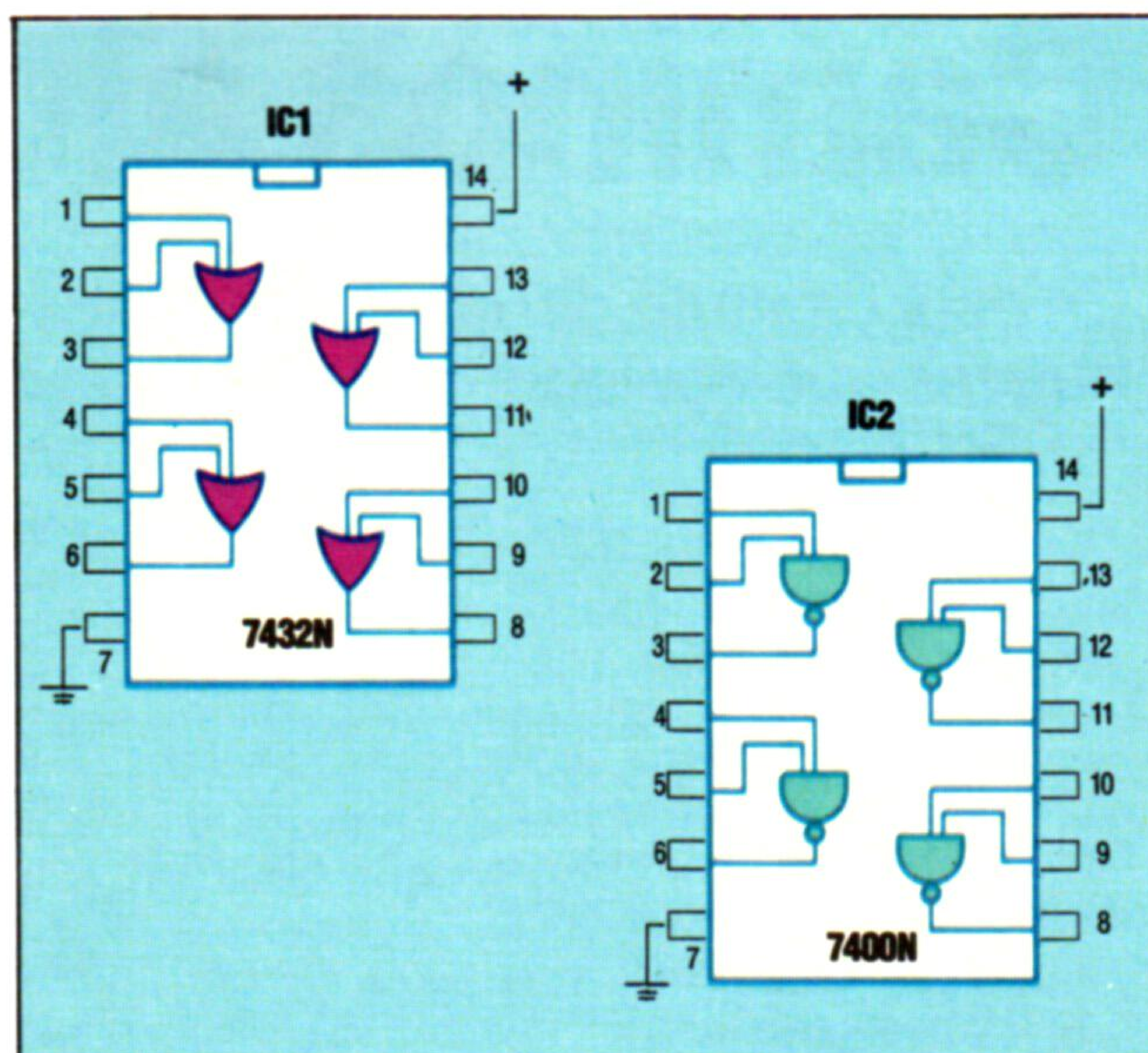
En la práctica, los chips rara vez se emplean de este modo y sólo ocasionalmente aparecen en un ángulo del circuito realizando una tarea de menor importancia. De los chips mayores salen y entran más señales, de manera que el chip en su totalidad es un dispositivo completo capaz, por ejemplo, de sumar dos números de cuatro bits.

El nivel de complejidad crece hasta tal punto que determinados chips son capaces de realizar por sí mismos tareas completas.



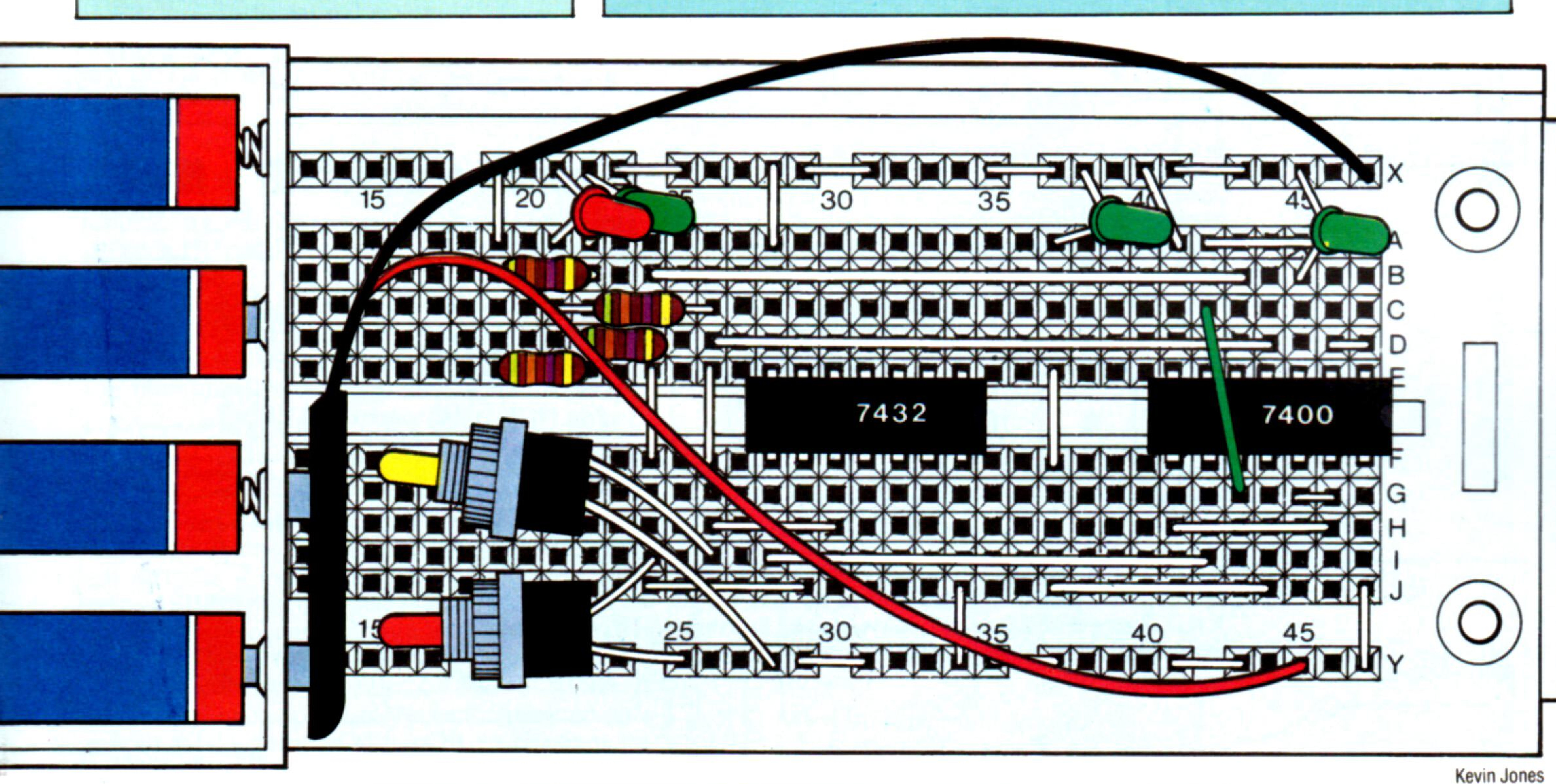
Materiales

4 resistencias de 500 ohmios, de 1/4 de vatio
4 diodos emisores de luz (LED)
2 conmutadores de funcionamiento por presión
1 circuito integrado 7400N
1 circuito integrado 7432N
4 pilas HP7 o equivalentes
1 soporte de pilas
1 conector para pilas
1 tablero de montaje de prueba
Trozos cortos de cable



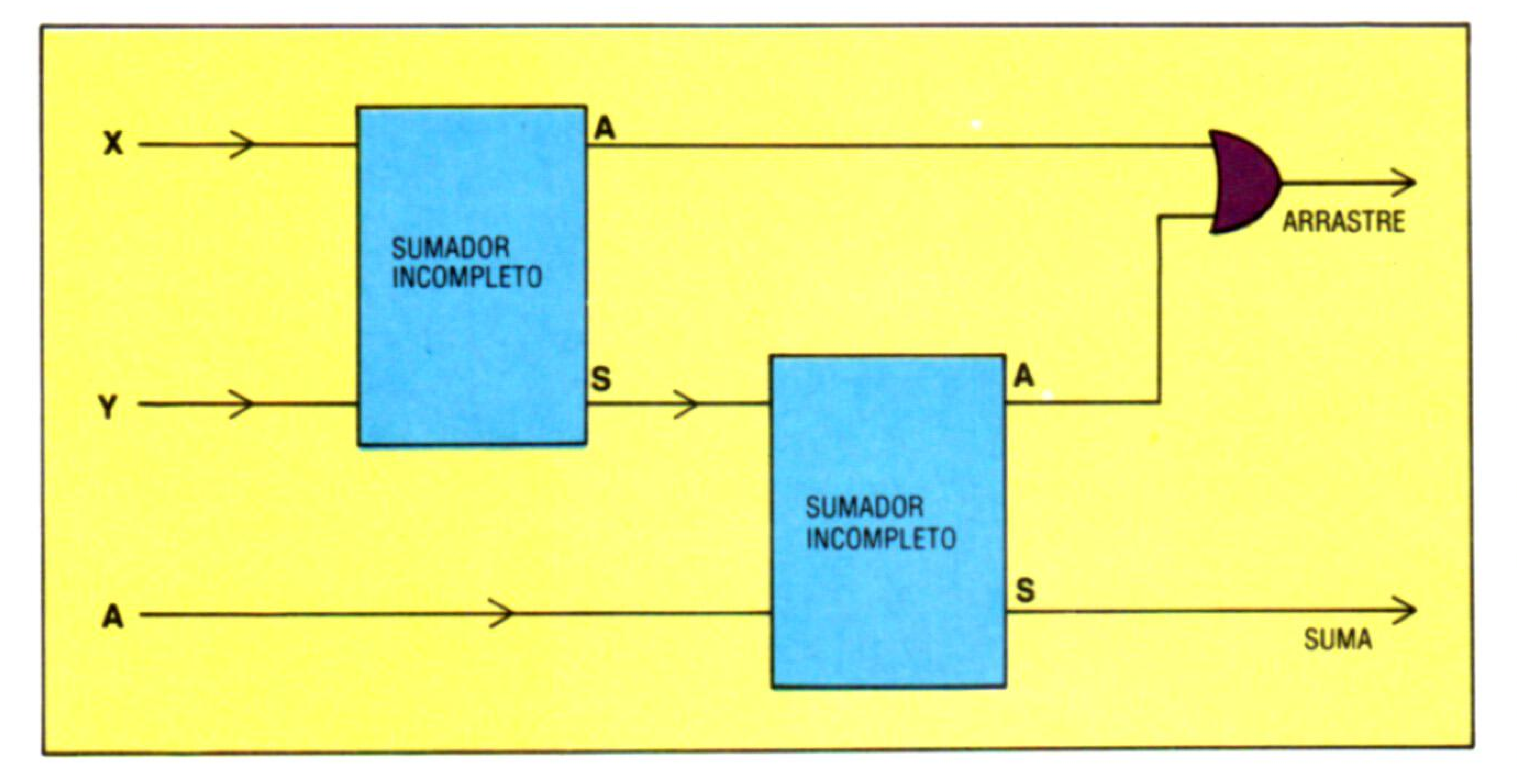
Patillas de salida

Los dos circuitos integrados que emplearemos son chips TTL de 14 patillas. El 7400N es cuatro puertas NAND en un chip y el 7432N ofrece cuatro puertas OR. Este diagrama indica el modo en que se disponen las puertas; lo único que queda por hacer es conectar las patillas adecuadas. En la parte superior del chip hay una muesca que le indica el modo de utilizarlo. Recuerde también que debe suministrar energía a través de las patillas 7 y 14 a fin de que los chips funcionen



Kevin Jones

A través del circuito
Una vez diseñado el circuito electrónico, el paso siguiente consiste en acomodar los componentes en el tablero de montaje de prueba. Puede adquirir tableros prefabricados o, simplemente, utilizar una fotocopia en un circuito vacío. Conviene que el tablero se parezca al máximo al circuito original ya que, cuanto más claro sea el diseño, más fácil resultará su construcción. Cópielo con exactitud pues todos los componentes están en la posición correcta



Sumador completo
¿Quiere ampliar como ejercicio su sumador incompleto convirtiéndolo en un sumador completo? Dicho circuito no sólo suma dos bits, sino también el arrastre de cualquier posición anterior de bit. Una serie de sumadores completos puede sumar palabras binarias completas. El modo más simple de crear un sumador completo consiste en construir dos sumadores incompletos como los que aquí se muestran. La señal suma del primer sumador incompleto debe reemplazar uno de los conmutadores de entrada del segundo sumador incompleto. La salida de arrastre del primer sumador incompleto debe ser pasada por OR junto con la del segundo para producir la señal del LED de arrastre

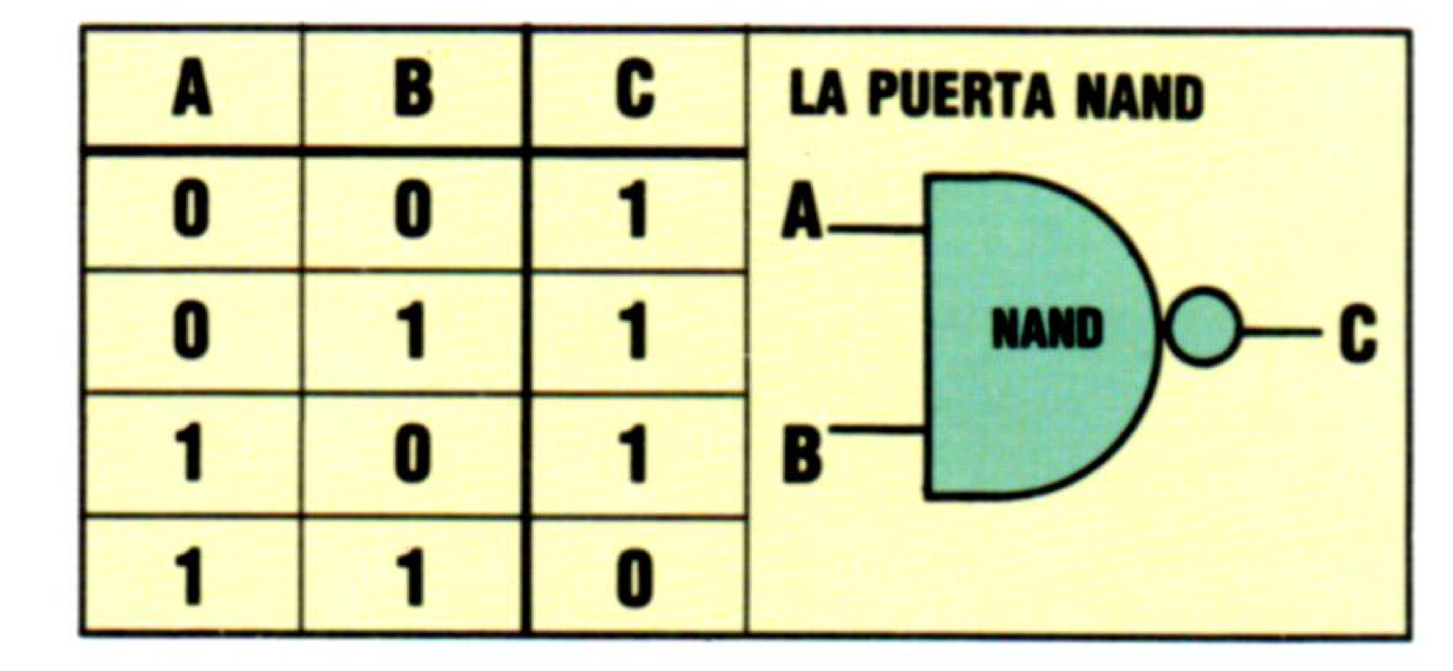
Caminos alternativos

En este capítulo del curso de lógica introducimos dos puertas nuevas que nos abren una ruta diferente para diseñar circuitos

Si es posible resolver todos los problemas lógicos utilizando puertas AND, OR y NOT, ¿para qué complicarnos la vida estudiando otros tipos de puertas? Porque con estas nuevas puertas podemos reducir el costo de fabricación del circuito. Es posible resolver todos los problemas lógicos utilizando alguna de las siguientes técnicas:

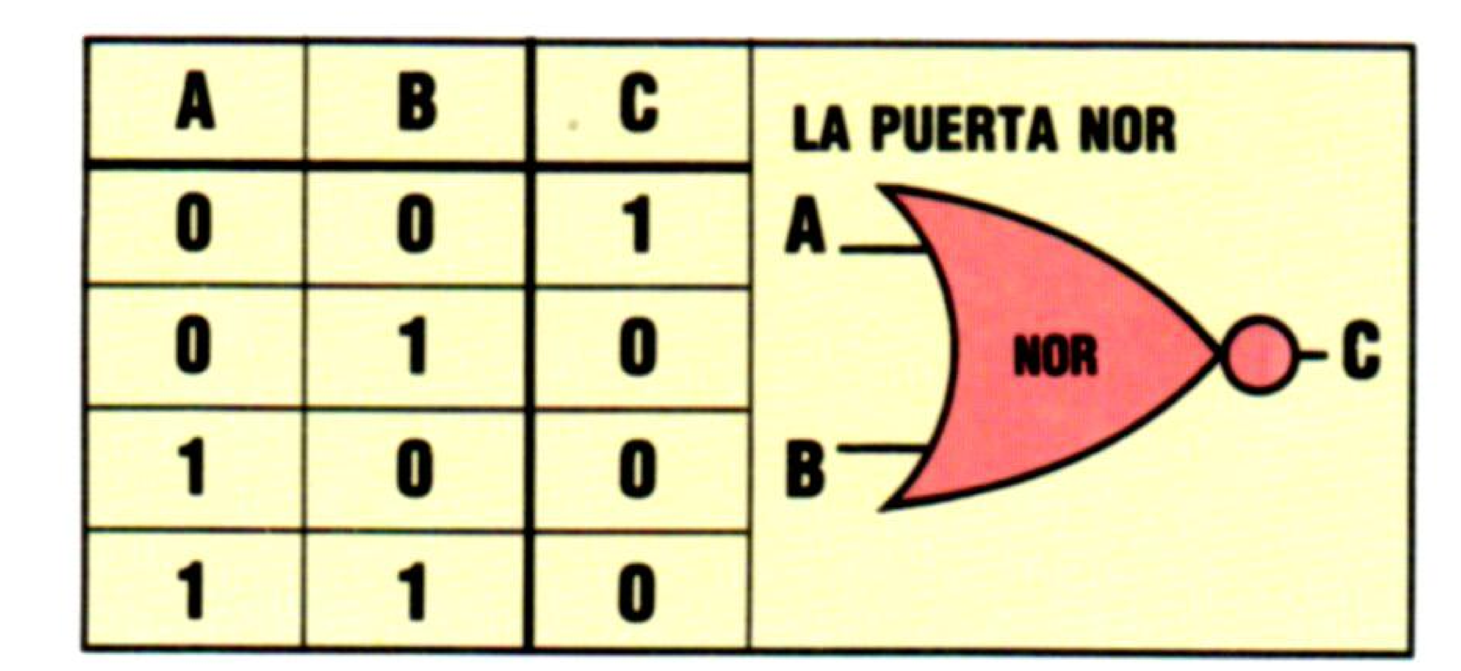
a) puertas AND, OR y NOT juntas
b) sólo puertas NAND
c) sólo puertas NOR
d) una combinación de todas ellas

Analicemos esos dos nuevos tipos de puertas. Como ocurre con todo circuito y sus elementos, la función de cada puerta queda mejor descrita por su tabla de verdad.



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

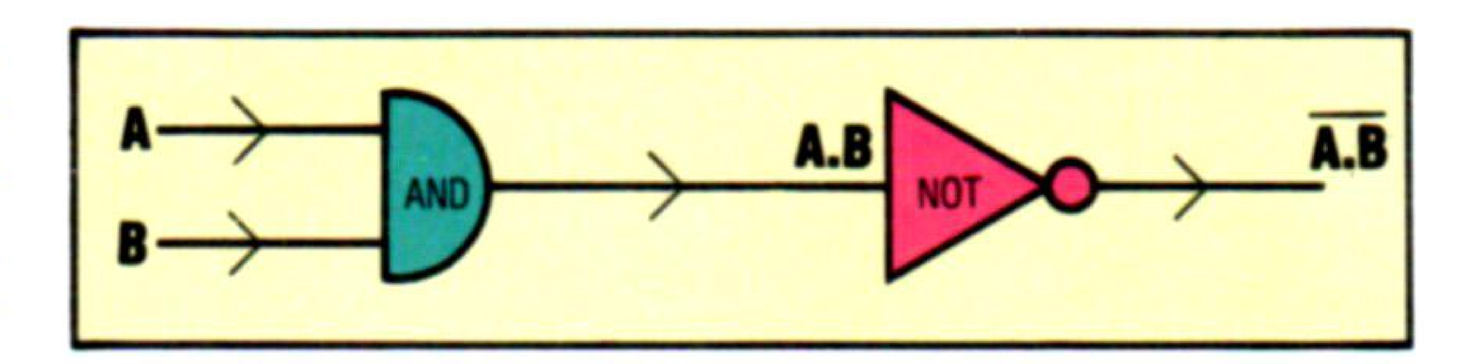
NAND es la abreviación de No AND y, si comparamos esta tabla de verdad con la de una puerta AND (véase p. 488), podemos ver que en la columna de salida todos los unos han sido cambiados por ceros y viceversa.



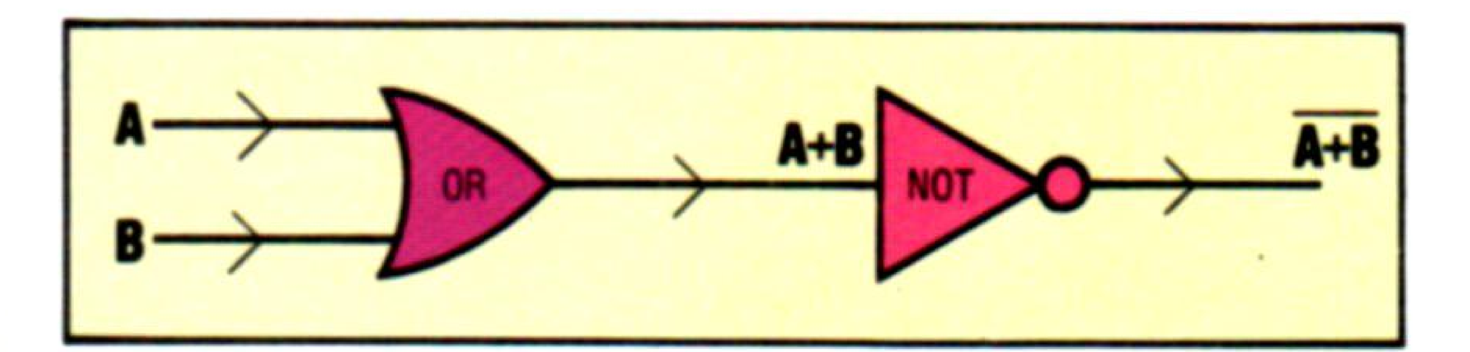
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

De manera semejante, NOR es apócope de No OR y la comparación de las columnas de salida de esta tabla con las de la tabla de una puerta OR (véase p. 488) vuelve a mostrar que todos los unos y los ceros han sido invertidos.

En álgebra booleana no existen símbolos especiales para las operaciones NAND y NOR, si bien podemos representar cada función utilizando los símbolos AND, OR y NOT que ya conocemos. Una puerta NAND equivale a este simple circuito:



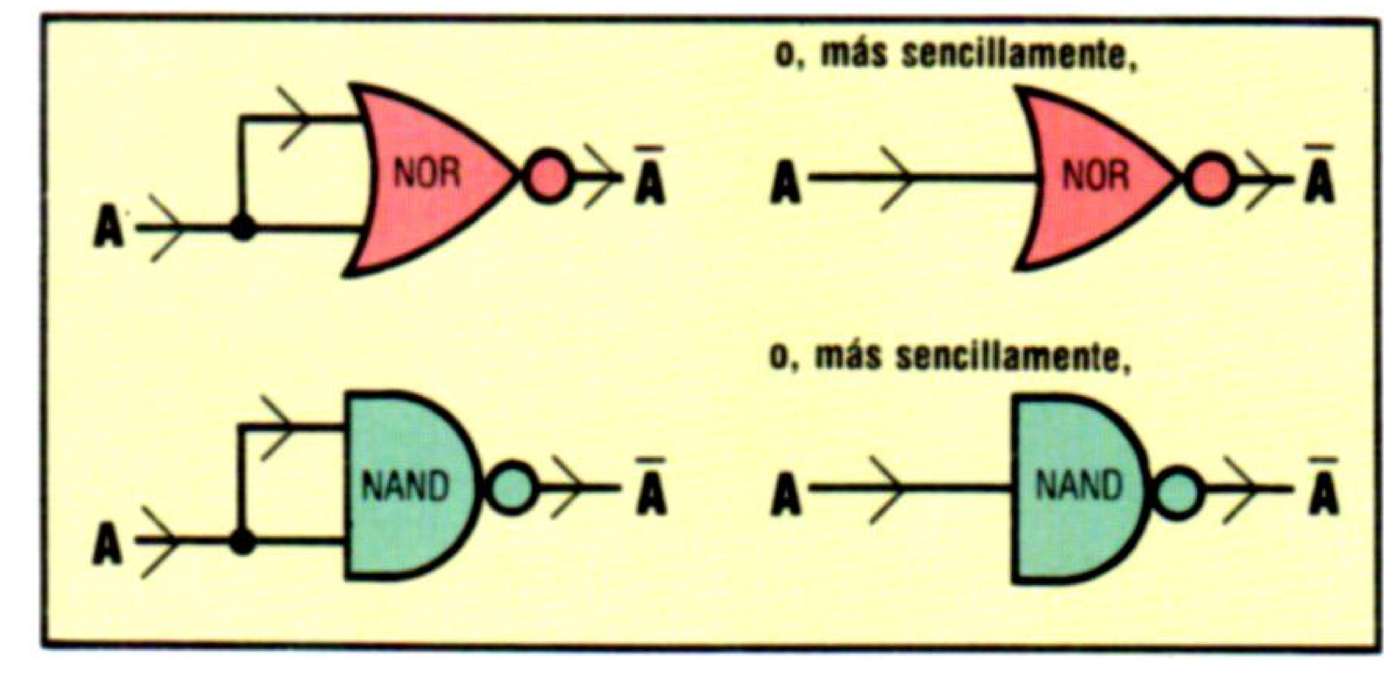
y la puerta NOR es equivalente a una puerta OR seguida de una puerta NOT:



El empleo de NAND y NOR

Del mismo modo que es posible dibujar circuitos AND/OR/NOT que equivalen a NAND y NOR, también podemos representar cada una de estas tres puertas básicas mediante una serie de puertas NOR o de una serie de puertas NAND.

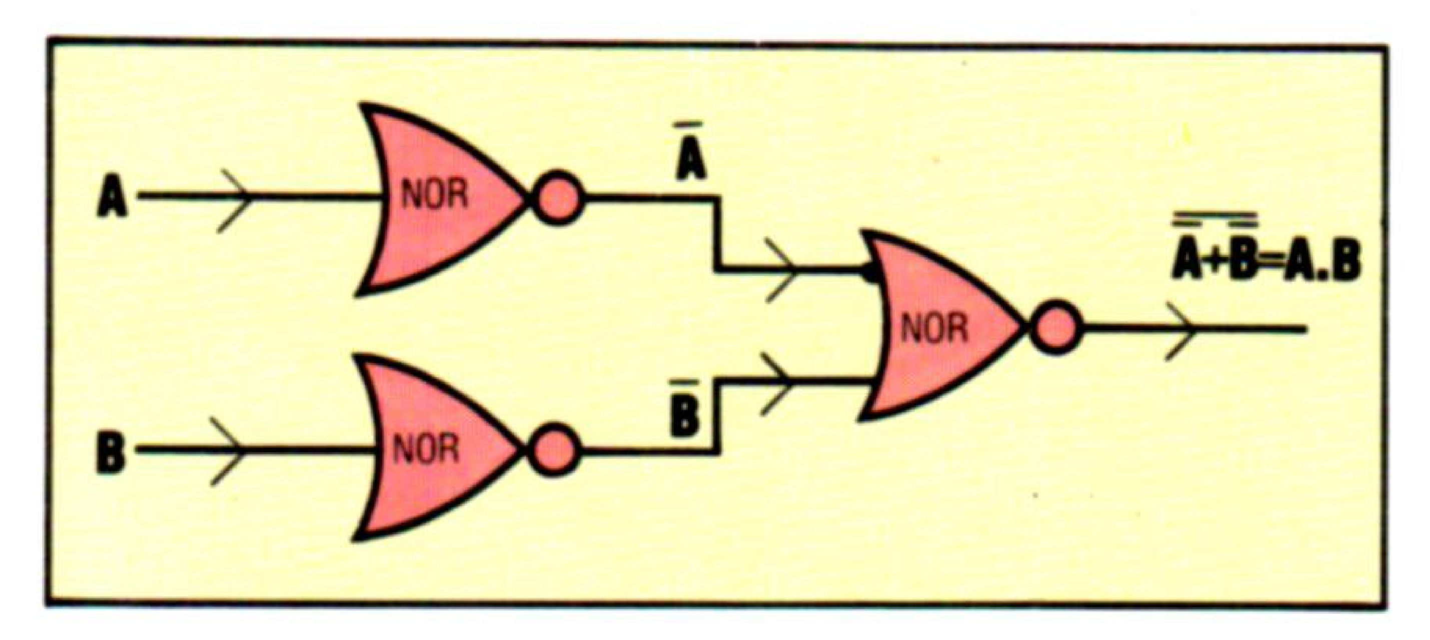
Puertas NOT: La anulación puede conseguirse conectando ambas entradas, ya sea utilizando una puerta NOR o una puerta NAND:



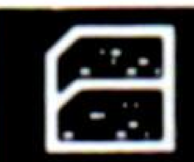
Puertas AND: En términos de álgebra booleana, la salida de una puerta AND cuyas entradas son A y B es A.B. No obstante, podemos manipular esta expresión hasta darle una forma más útil.

$$A.B = \overline{\overline{A}}.\overline{\overline{B}} \quad \text{(puesto que } A = \overline{\overline{A}}\text{)}$$
$$= \overline{\overline{A} + \overline{B}} \quad \text{(ley de Morgan)}$$

Así, es posible hacer el circuito poniendo NOT(A) y NOT(B) a través de una puerta NOR:



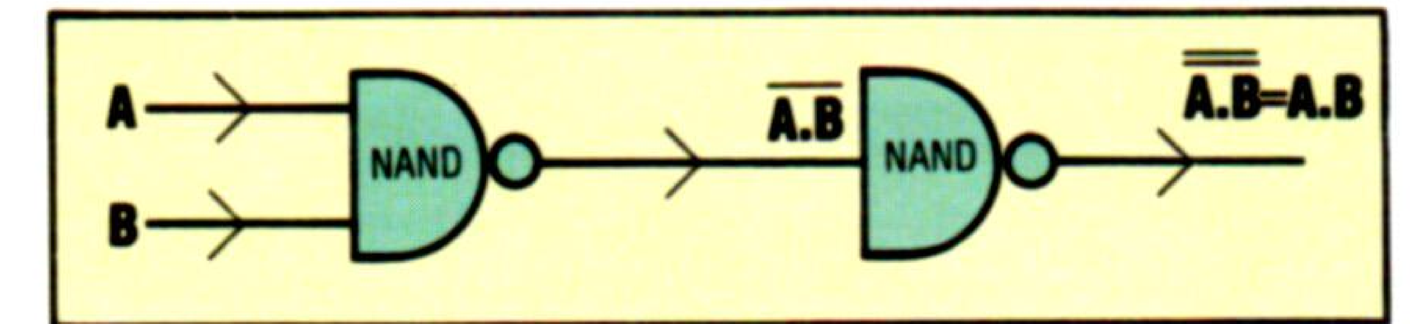
También es posible crear una puerta AND utilizando puertas NAND. La salida de una puerta NAND



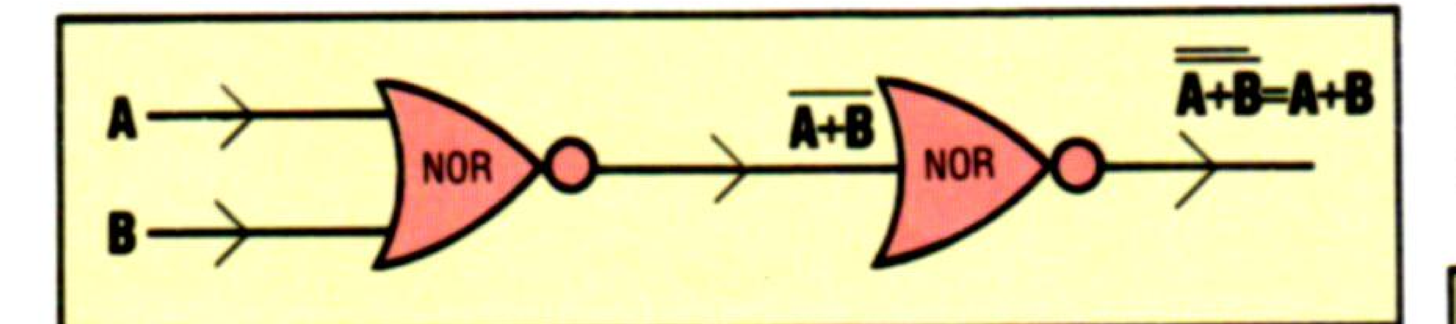
es $\overline{A.B}$. Si esta salida es invertida, obtendremos:

$\overline{\overline{A.B}} = A.B$

En consecuencia, el circuito será:



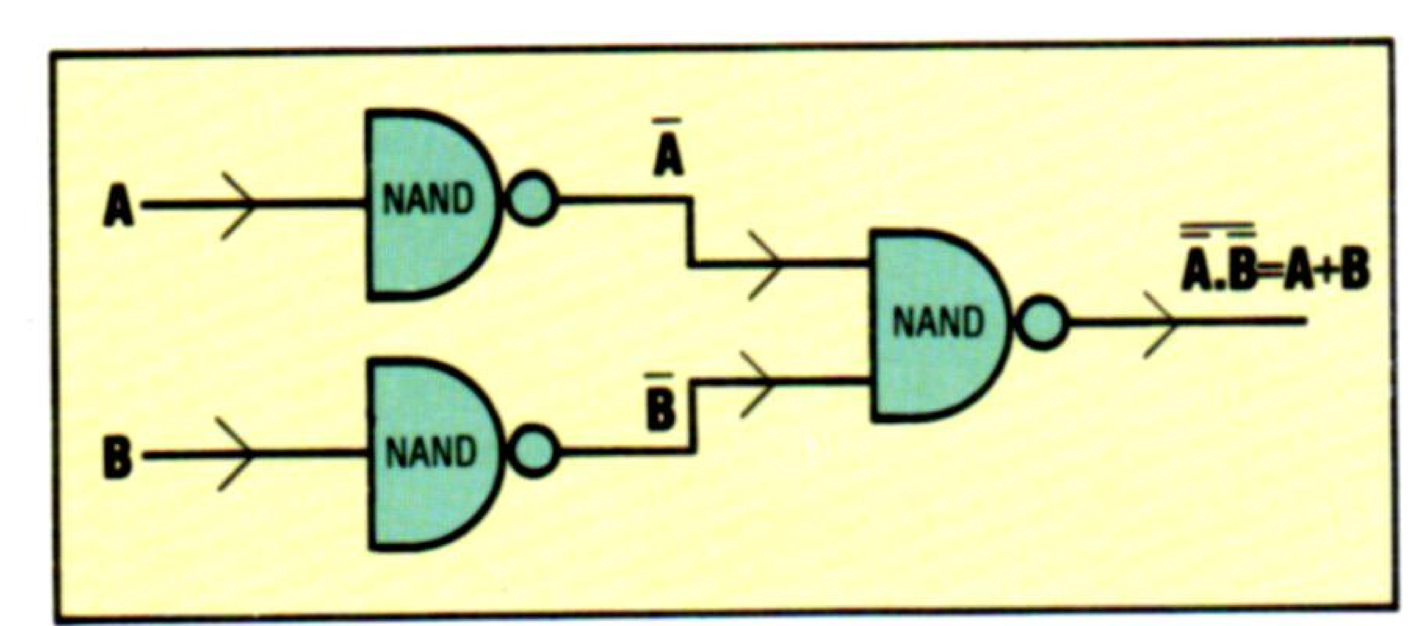
Puertas OR: Del mismo modo que enlazar dos puertas NAND equivale a una puerta AND, si enlazamos dos puertas NOR obtendremos un circuito que es equivalente a una puerta OR:



La salida exigida de una puerta OR es A + B. Recurriendo a las reglas del álgebra booleana, podemos manipularla hasta darle forma característica de una puerta NAND:

$$A + B = \overline{\overline{A}} + \overline{\overline{B}} = \overline{\overline{A}.\overline{B}}$$

y, por lo tanto, el circuito correspondiente con puertas NAND es:

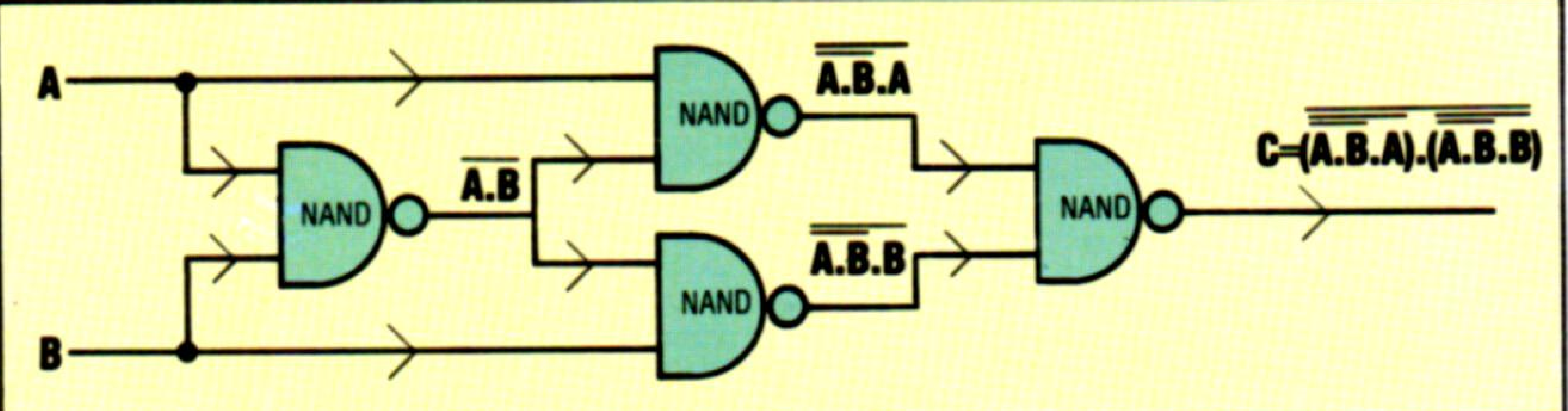


Si queremos construir un circuito utilizando únicamente elementos NAND o NOR, podemos seguir los métodos de simplificación ya conocidos, pero primero debemos manipular la expresión de álgebra booleana definitiva hasta darle la forma adecuada. Para circuitos que incorporan puertas NAND, aplicamos las reglas del álgebra booleana hasta crear una expresión que se compone de grupos de AND conectados por OR y utilizamos las leyes de Morgan tantas veces como sea necesario hasta que la expresión quede totalmente en forma de NAND. Para circuitos en forma de NOR, empleamos las mismas reglas, según muestra el ejemplo. Para ver cómo se utilizan dichas reglas, hagamos un repaso de la puerta OR Exclusiva (XOR), que aparece en la página 527. La salida de una puerta XOR puede definirse mediante la expresión $C = \overline{A.B}.(A + B)$.

Tomemos esta expresión y convirtámosla de modo que podamos construir un circuito para la puerta XOR exclusivamente con puertas NAND. En primer lugar, manipularemos la expresión hasta obtener grupos de AND conectados por OR.

$$C = \overline{A.B}.(A + B)$$
$$= \overline{\overline{(\overline{A.B}.A) + (\overline{A.B}.B)}} \text{ (prop. distributiva)}$$
$$= \overline{\overline{(\overline{A.B}.A)}.\overline{(\overline{A.B}.B)}} \text{ (ley de Morgan)}$$

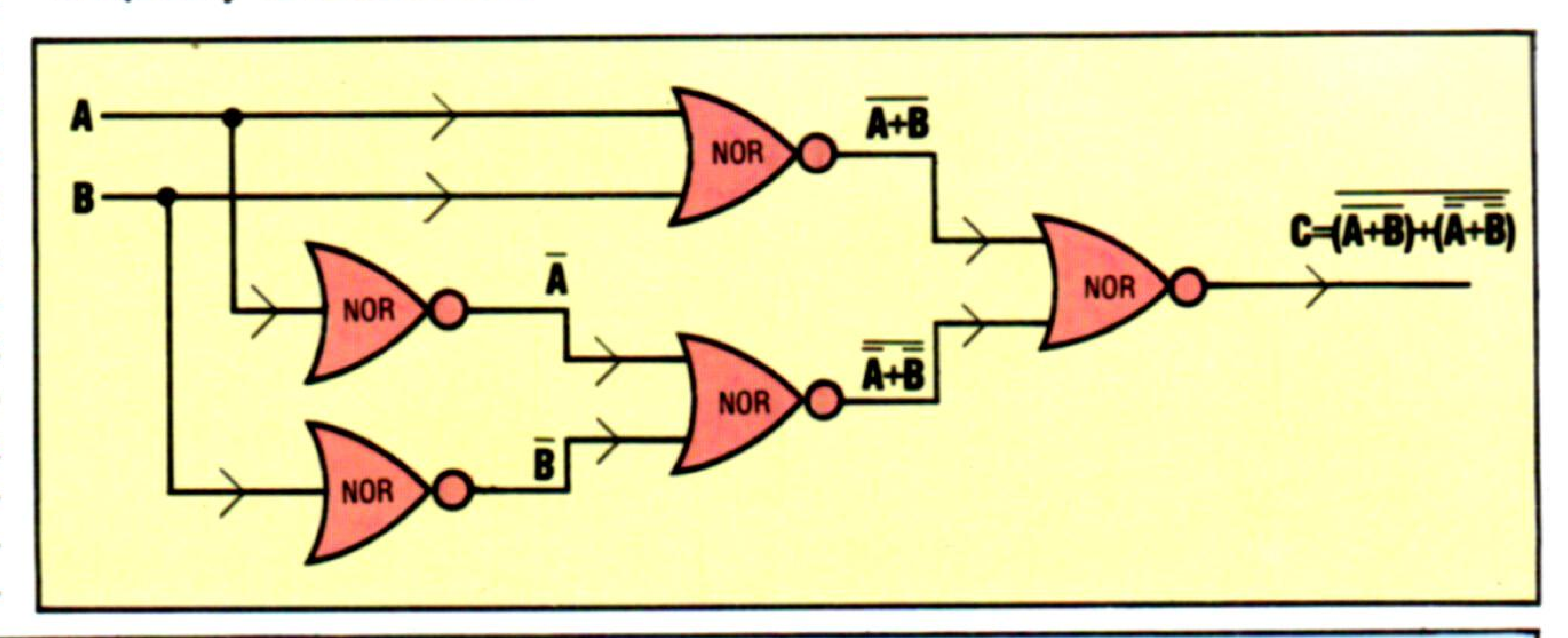
Cuando se dibuja el circuito de una expresión tan complicada como ésta, conviene comenzar por la salida y retroceder hasta las entradas. Procure seguir este diagrama de circuito desde la salida para ver cómo se construyó.



En cuanto a la forma NOR, debemos comenzar una vez más con la expresión original simplificada de la puerta XOR y manipularla formando grupos de OR conectados por AND. Este primer paso puede darse aplicando la ley de Morgan a la parte izquierda de la expresión:

$$C = \overline{A.B}.(A + B)$$
$$= \overline{\overline{(\overline{A} + \overline{B}).(A + B)}}$$
$$= \overline{\overline{(\overline{A} + \overline{B})} + \overline{(A + B)}}$$

También en este caso, esta expresión se convierte mejor en un diagrama de circuito comenzando por la salida y retrocediendo.



Solución al ejercicio 6 de la p. 627

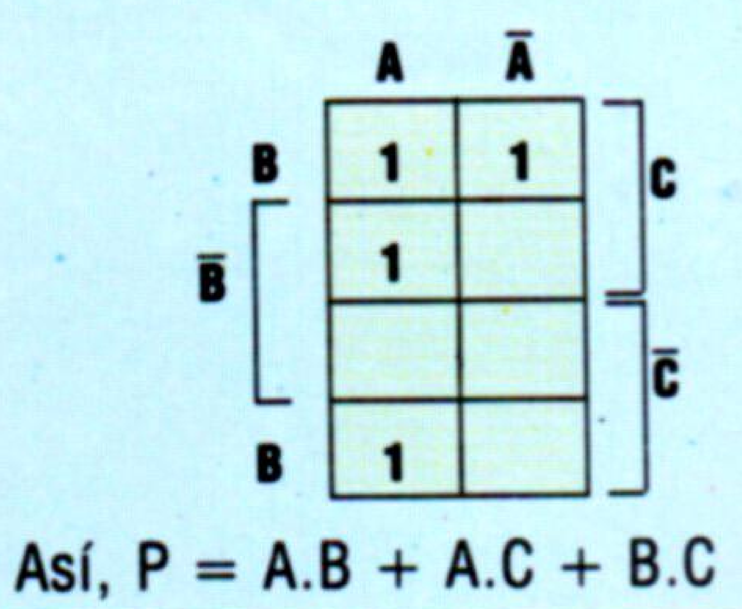
1a)

Entradas			Salida
A	B	C	P
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

b) La expresión booleana de P es:

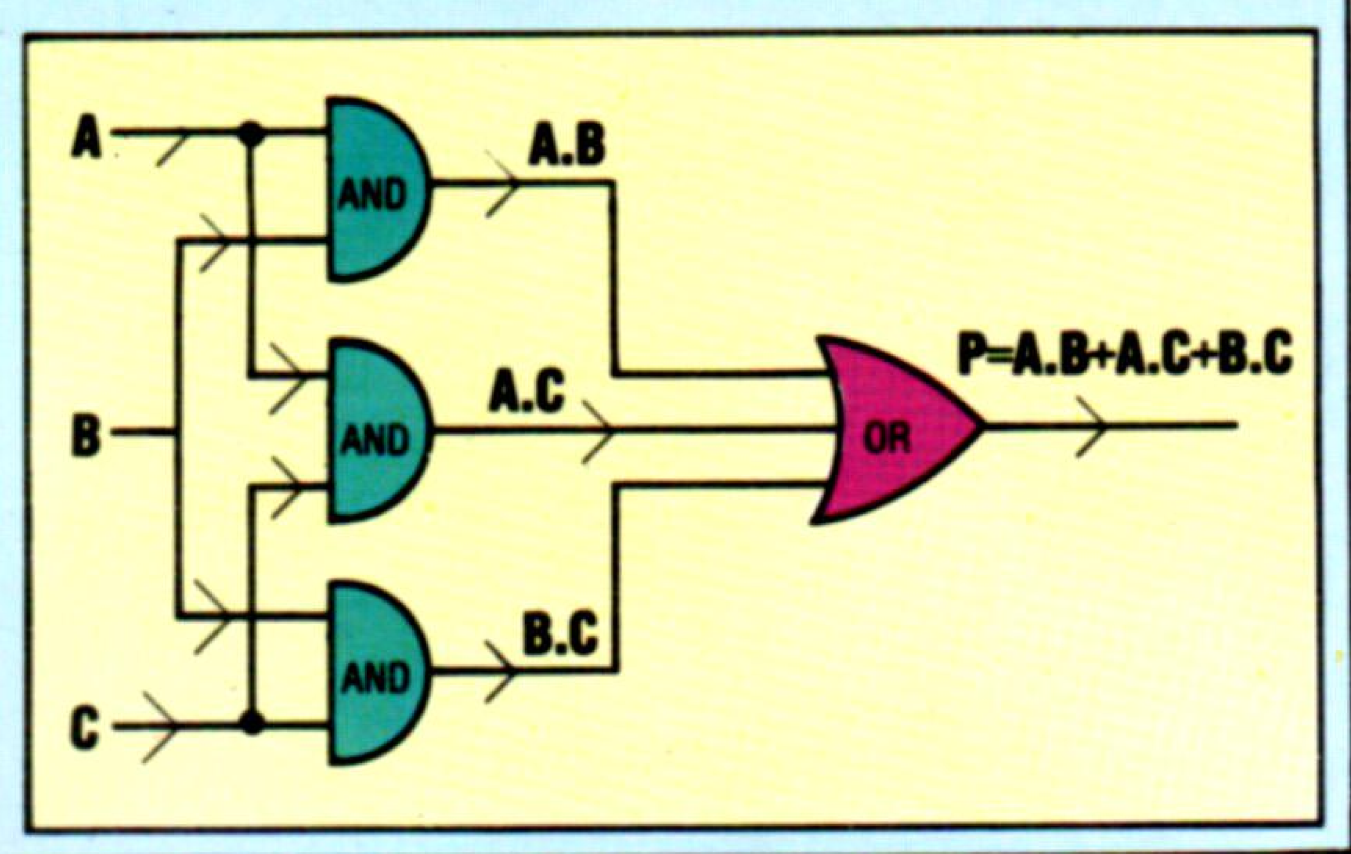
$P = \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$

Puede simplificarse utilizando un diagrama de Karnaugh:



Así, P = A.B + A.C + B.C

c) El circuito es:



Liz Dixon

Conectores

Hay dos símbolos que, en los diagramas largos y complejos, resultan imprescindibles

Figura 1

Estos símbolos se denominan *conector de partes* y *conector de páginas*. Un ejemplo ilustrará el uso del primero: un operario, situado al final de una línea de producción, recibe dos clases de contenedores —A y B—, con un número indeterminado de pernos. Su cometido en la cadena se reduce a lo siguiente: tomará un contenedor y, luego de comprobar si pertenece a la clase A, debe pesarlo, anotar su peso, embalarlo y expedirlo. En cambio, si el contenedor es de clase B, tomará un perno, lo pesará, calibrará y anotará sus medidas. Así ha de continuar, un perno tras otro, hasta agotar su número, llegado el último de los cuales pesará el contenedor, anotará su peso, lo embalará y, tras expedirlo, proseguirá con el siguiente.

Obsérvese en el ejemplo la utilización de los símbolos conectores de partes (fig. 1) del diagrama. Éstos realizan la misma función que las líneas del flujo, con la ventaja de que, conforme el ordinograma va creciendo, el número de líneas empleadas aumenta, pudiéndose dar el caso de que la profusión de éstas, con sus posibles cruces, llegue a inducir a error. Mediante estos pequeños círculos, puede accederse igualmente al punto marcado por el conector de entrada, cuya contraseña interior (una letra o número, preferentemente) coincide con la figura colocada en el conector de salida. El conector que marca la entrada a la secuencia debe ser único, es decir, no pueden existir dos conectores de entrada con una misma clave. Pero no hay límite para el número de conectores de salida.

Equiparando el caso al de las líneas de flujo, se pueden utilizar tantos conectores como sea conveniente en un mismo diagrama, y es posible que se alternen indistintamente con líneas de flujo en el mencionado ordinograma, pues no existe regla alguna que lo impida.

El conector de páginas (fig. 2), en cambio, es de carácter obligatorio siempre que un mismo ordinograma ocupe más de una hoja, figurando como contraseña el número de la página a que se refiere.

Llegados a este punto, puede comprobarse la gran ventaja, en cuanto a comodidad y claridad se refiere, que representa utilizar un conector en lugar de una línea de flujo para conectar puntos del diagrama representados en diferentes páginas.

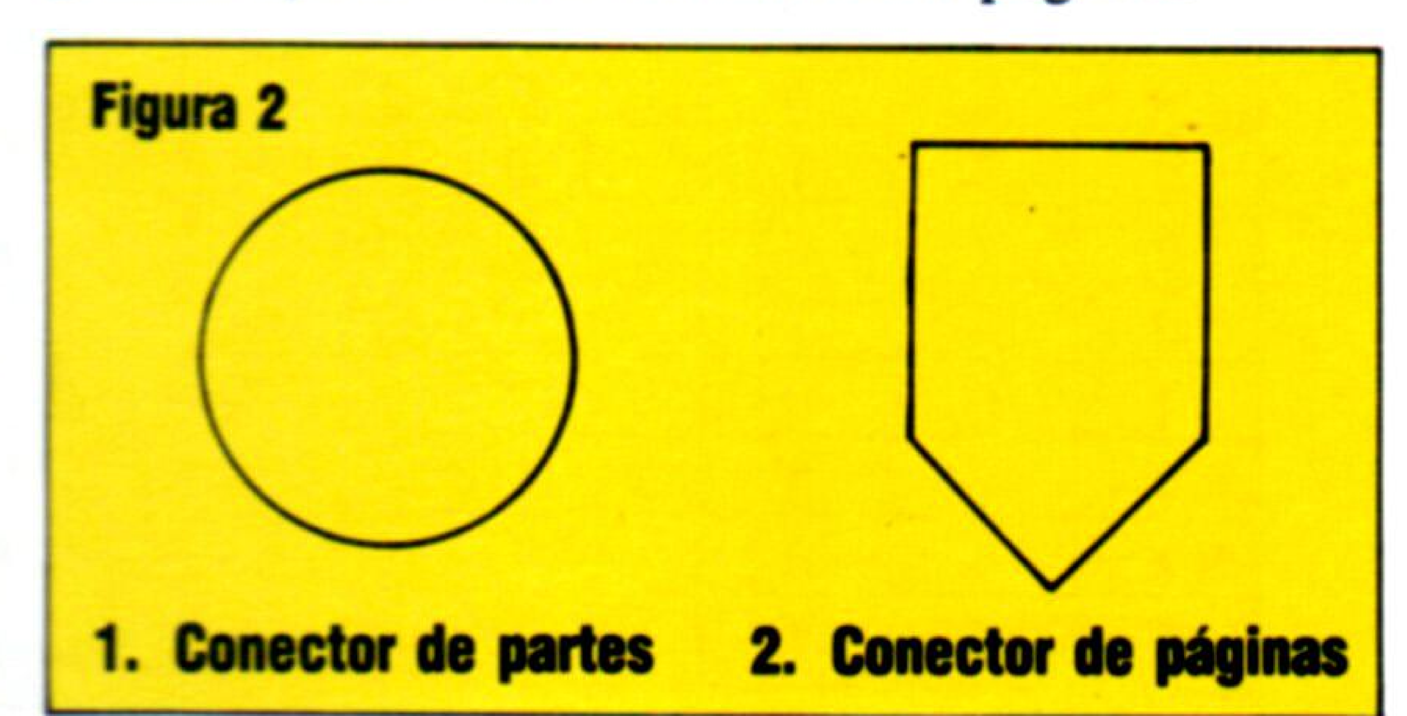


Figura 2

1. Conector de partes 2. Conector de páginas



Pizarra electrónica

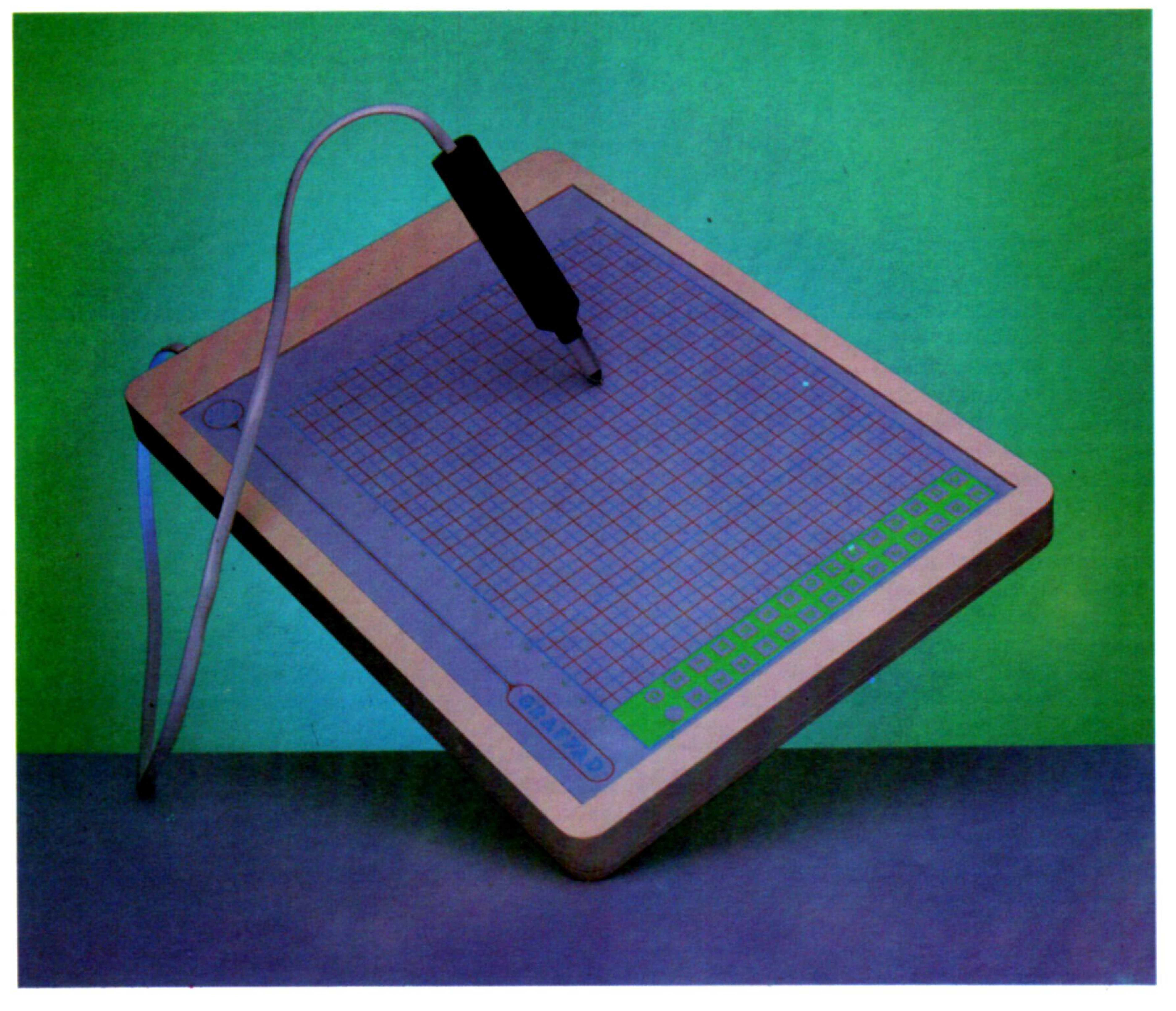
El Grafpad es un tablero de gráficos que permite crear detallados diseños y dibujos en un microordenador

Los tableros de gráficos, o digitalizadores, constituyen uno de los periféricos más polifacéticos y útiles de los microordenadores. Es evidente su uso como ayuda al dibujo y al diseño, desde el dibujo artístico al diseño de circuitos electrónicos y el trazado de mapas. Además de sus aplicaciones directas para el dibujo, ofrecen un útil dispositivo adicional de entrada. Una ficha colocada sobre el tablero gráfico puede tener todas las características de un programa, ya sea en palabras o gráficamente. Basta con tocar la orden apropiada con el cursor (o lápiz), y el software expondrá la opción escogida.

Estos sistemas solían ser coto cerrado de máquinas específicas, exclusivamente al alcance de diseñadores e ingenieros. Sin embargo, los precios han disminuido lo bastante para que los usuarios de ordenadores personales puedan probar por sí mismos este accesorio. La Grafpad, que aquí analizamos, es uno de los principales digitalizadores de bajo costo y ofrece buenas especificaciones por un precio razonable. Existen versiones exclusivas para el BBC Micro, Commodore 64 y el Spectrum de Sinclair. La que ilustramos corresponde a la versión para el BBC.

El Grafpad consta de tres elementos: la tablilla propiamente dicha, un cursor conectado y el software de control. La tablilla se conecta con el BBC a través de la puerta para usuario y el cursor se conecta a su lado. La superficie de la tablilla se presenta cuadriculada con 16 por 20 casillas y tiene una barra de mando (un listón aparte en el que se han inscrito letras individuales). La barra de mando puede utilizarse para controlar parte del software sin necesidad de recurrir a un teclado. Encima de ésta se extiende una cubierta de plexiglás que protege la superficie de la tablilla. Es posible diseñar los propios segmentos de programas (*overlays*) mediante las órdenes y rejillas que usted desee.

Dentro de la tablilla hay una rejilla de 320 por 256 cables, separados aproximadamente por una distancia de 1,2 mm. La punta del cursor es un minúsculo interruptor. Cuando se presiona con el cur-



Ideas gráficas
El Grafpad puede utilizarse con su propio software para crear diseños y dibujos o con sus programas como dispositivo de entrada

Máquina sumadora

He aquí la versión inglesa de un programa en el BBC Micro que utiliza el Grafpad como dispositivo de entrada para una sumadora. Bajo la cubierta de la tablilla se sitúa un segmento de programa con las claves de la sumadora. Al tocar la clave pertinente con el cursor, el software se pondrá en operación

```
10 REM GRAFPAD DEMO
20 REM
30 REM An adding machine using the Grafpad
40 REM
50 MODE 1
60 PRINT "ADDING MACHINE":PRINT
70 HIMEM=&29FF
80 XADJUST=7
90 REM
100 REM set up co-ordinate table
110 REM
120 DIM B(15):FOR I=1 TO 15:B(I)=I*25:NEXT I
130 R1=0:R=0
140 REM
150 REM load Grafpad driver program
160 REM
170 *LOAD "PADREAD" 2A00
180 PEN%=&2A00
190 REM
200 REM MAIN PROGRAM STARTS HERE
210 REM
220 REM wait for pen to be pressed
230 CALL PEN%
240 !X%=!X%-XADJUST:IF !X%<0 THEN !X%=0
250 IF ?U%>0 THEN 230
260 REM beep to register pen press
270 SOUND 1,-15,120,1
280 X=!X%
290 REM convert position to 0-12
300 I=0
310 IF X>B(I) THEN I=I+1:GOTO 310
320 REM interpret codes 0-12
330 IF I<10 THEN R=R*10+I
340 IF I=10 THEN PRINT R;" +":R1=R1+R:R=0
350 IF I=11 THEN PRINT R;" =":PRINT "        ____":R1=
R1+R:PRINT R1:R1=0:R=0:PRINT:PRINT
360 IF I=12 THEN PRINT:PRINT"CLEAR":PRINT:R=0:R1=0
370 REM wait until pen is lifted again
380 CALL PEN%
390 IF ?U%=0 THEN 380
400 REM loop for next pen press
410 GOTO 230
```

Ø 1 2 3 4 5 6 7 8 9 + = Clear

```
ADDING MACHINE

  34 +       4 +
  96 =       8 =
 ----      ----
 130        12

   3 +   19876 +
   9 +    8876 +
   6 +    8864 +
  28 +    1243 =
 126 =   -----
 ----    38859
 172
```

Superficie de dibujo
La superficie se divide en 20 columnas de 16 casillas. Puede detectar el cursor (o lápiz) en cualquiera de las 320 por 256 posiciones

Interface
El Grafpad se conecta en el enchufe de ampliación al bus de 1 MHz del BBC

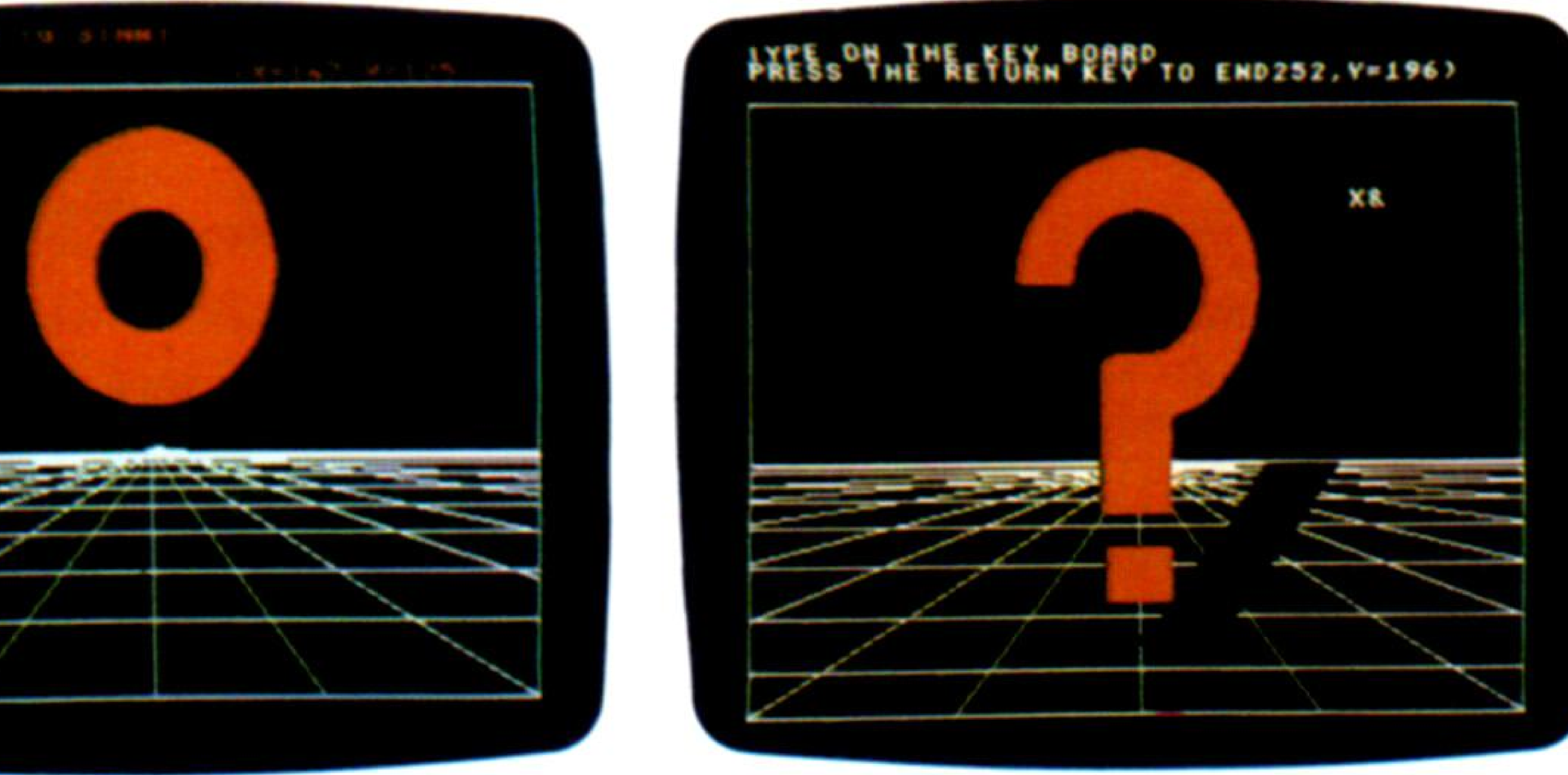
Sistema de circuitos
Cuando el cursor se posa en la superficie, una ULA explora las filas y las columnas, buscando un cambio en la capacidad que le permita detectar su posición

sor la cubierta de plexiglás de la tablilla, un chip ULA (*Uncommitted Logic Array*: disposición lógica no comprometida) pulsa cada uno de los alambres hasta detectar la posición del lápiz mediante un cambio en la capacidad. Esta exploración tiene lugar 2 000 veces por segundo, lo que convierte la localización del cursor en un proceso muy rápido. Para ayudar a que el sistema funcione de manera fiable, conviene sostener el cursor por la banda metálica que rodea la punta.

Cuando se posa el lápiz en la superficie, el ordenador recibe la señal de "cursor en acción" y un informe de sus coordenadas en la tablilla. El efecto exacto que crea queda determinado por el software. En la pantalla puede aparecer un cursor en forma de cruz en la posición correspondiente, o puede desencadenarse una orden específica. Es aquí donde empieza a notarse la economía de la Grafpad. El lápiz sólo puede detectarse en una rejilla de 320 por 256 posiciones, lo que vuelve muy difícil dibujar detalles muy tenues o finos. Asimismo, la tablilla es muy pequeña: una hoja de papel Din A-4 es una zona de trabajo muy sensible.

El Grafpad dispone de tres paquetes de software, que van desde una simple rutina de demostración, pasando por un sencillo programa de dibujo,







Software de diseñador
El Grafpad viene con PROG2, un paquete de dibujo a pulso que se utilizó para crear estas imágenes. La capacidad para dibujar y rellenar círculos abrevia en gran medida el proceso

Cursor
Cada vez que el cursor se pone en contacto con la tablilla, su posición es transmitida al ordenador

Rejilla
Se utiliza una matriz de cables separados por 1,2 mm para determinar la posición del lápiz en la tablilla

Interruptor
El cursor del Grafpad tiene incorporado en la punta un minúsculo interruptor que desencadena la búsqueda de la posición del mismo en la plantilla

Barra de mando
Una zona concreta de la superficie presenta un conjunto de letras y de órdenes numéricas que puede emplearse con determinados programas

Capa de plexiglás
Una lámina de plexiglás protege la parte superior de la tablilla. A ésta se le pueden adherir hojas de superposición

hasta un complejo paquete de CAD (*Computer-Aided Design*: diseño auxiliado por ordenador). Como se suministra en lenguaje máquina y en una versión de BASIC, es posible incorporar a estos programas la rutina simple de leer la superficie.

El programa de dibujo es un programa electrónico comparable a la mayoría de los paquetes artísticos existentes, incluso a aquellos que no cuentan con tablilla. Presenta todas las características básicas: líneas, casillas, círculos, triángulos y "dibujo a pulso", y puede rellenar una zona específica con determinado color. No obstante, carece de posibilidades más complejas como poder copiar y trasladar secciones del dibujo. A decir verdad, no tiene nada que un software para teclado no pueda hacer, si bien el Grafpad permite trazar diseños. La versión para el BBC sólo ofrece cuatro colores simultáneos y presenta la desventaja de que sus tiempos de respuesta son lentos.

El programa CAD es, lisa y llanamente, una demostración de algunos de los principios en juego. En primer lugar, usted crea una serie de caracteres que utilizará en la construcción de los diseños. En el caso de la electrónica, dichas formas pueden ser componentes como transistores, resistencias, etc. También puede crear puertas lógicas, diseños de muebles y hasta dibujos de azulejos. Una vez creados, usted se traslada al tablero de dibujo real donde puede repetir y acomodar libremente las formas e incorporarles líneas rectas.

Esto es muy parecido al funcionamiento de un auténtico paquete de CAD. Sin embargo, el software de la Grafpad no se plantea un uso serio. Entre los medios que necesitaría figuran la capacidad de etiquetar los diagramas, rotar los dibujos y ponerlos a escala, ampliar un fragmento específico de la pantalla, situar con toda exactitud formas pequeñas, etc. Es básica una mayor flexibilidad para corregir los errores y, en un sentido general, el programa CAD no permite utilizar el Grafpad como dispositivo de entrada. Pese a la pequeña barra de mando, muchas órdenes requieren entrada por el teclado y la operación general resulta pesada.

El Grafpad propiamente dicho es un periférico polifacético que ofrece mucho a cambio de lo que cuesta. Su superficie, resolución y fiabilidad son restringidas a fin de que su precio sea bajo. Sin embargo, el software que acompaña al sistema resulta decepcionante y la unidad atraerá más que a nadie a aquellos que quieran escribir sus propios programas. Pese a todo, mediante el esfuerzo adecuado, los tableros de gráficos como éste permitirán que los usuarios estudien nuevas posibilidades y se convertirán en un estímulo considerable para el diseño gráfico más avanzado de los micros personales.

Hacer inventario

Prosiguiendo con nuestra serie sobre informática contable, analizamos ahora cómo se pueden controlar eficazmente los envíos y pedidos de mercancías

En una empresa perfectamente organizada, donde el propietario o el gerente están al corriente de la demanda de sus clientes y el género de que disponen, pocas veces se producirá exceso o falta de existencias. Estas dos últimas situaciones son consecuencia de una información deficiente. Y los sistemas computerizados de almacén son un modo excelente de evitar una información insatisfactoria.

Para cumplir con la tarea del control de almacén, los ordenadores tienen que proporcionar varias informaciones a gerencia. La empresa necesita saber el volumen de existencias, la lenta o rápida rotación que caracteriza a determinados artículos, en qué momento debe procederse a una reorganización, así como el valor de lo que hay en existencia.

El sistema se propone controlar sus movimientos. Éstos pueden descomponerse en las siguientes categorías: salidas de género enviado a tenor de los pedidos; entradas de género recibido de los proveedores; género preparado para cubrir pedidos y género en curso.

A estas cuatro categorías hay que añadir otro tipo de movimiento según las devoluciones de género realizadas por los clientes o por la empresa a sus proveedores. A menudo el inventario también presenta discrepancias respecto de lo que hay realmente y de lo que tendría que haber en las estanterías.

El sistema debe valorar también las existencias. Por ello, además de registrar cantidades y controlar los movimientos de género, el programa tiene que dar información sobre precios.

Los sistemas de control de almacén se dividen en dos clases bastante diferenciadas entre sí, según que su destino sean pequeñas empresas de venta al por menor y cadenas de distribución, o bien sean fábricas. En este segundo caso, el sistema de almacén ha de tener en cuenta ciertas salidas de almacén que se incorporan al proceso de fabricación y vuelven a él en forma de unidad producida. Muchos sistemas de control de existencias basados en el microordenador intentan satisfacer las necesidades de ambos tipos de empresa. En este capítulo trataremos sólo de las empresas de venta al por menor.

Mantener las estanterías llenas
Las cajas registradoras automatizadas pueden extraer directamente la información sobre los productos de las etiquetas con código de barras y registrar las ventas en un ordenador central de control de existencias. Esta realimentación instantánea permite a las grandes tiendas tener la seguridad de que las estanterías y los almacenes contienen los productos pertinentes en las cantidades adecuadas

Cortesía de J. Sainsbury Plc.

Puesto que el control de almacén está relacionado con tantos aspectos de las actividades de una empresa, es corriente que los sistemas de existencia *integren* una serie de programas ("integración" significa que dos o más paquetes de aplicación son capaces de intercambiarse valores y datos). Un sistema plenamente integrado incluiría, por ejemplo, un libro de compras, un sistema para procesar pedidos de compra, un módulo para facturación, un libro de ventas y un procesador de pedidos.

La integración reporta varias ventajas. Tomemos, por ejemplo, una empresa que ha integrado su sistema de control de almacén con un sistema para procesar pedidos de ventas. Si ambos sistemas están en condiciones de comunicarse entre sí, es posible actualizar automáticamente los archivos de existencias al tiempo que se procesa el pedido. Y, si el sistema que procesa las ventas puede acceder al archivo de existencias para obtener una completa descripción del artículo y su precio de venta, con sólo introducir el código de existencias, el operador tendrá que manejar menos datos... y tendrá menos oportunidades de hacer entradas erróneas.

El punto de partida de todo sistema de control de almacén es, obviamente, el archivo de datos de existencias. Cada sistema tendrá un modo de identificar todos los artículos mediante un código numérico específico y un texto descriptivo. El programa utiliza ese número como clave de archivo.

Se trata de un sistema de almacén relativamente simple pero la adición de otros paquetes más sofisticados lo convierte en un potente instrumento. El *Stock recording system* de Dragon Data para el Dragon 64, provisto de unidad de disco flexible, es un ejemplo del sistema más sencillo.

Este paquete ofrece al usuario ocho caracteres alfanuméricos para codificar el artículo, más un código de grupo de productos de dos dígitos. Ello significa que cualquier artículo puede incluirse en uno de los 50 grupos de productos (el máximo que ofrece el sistema). Si a cualquier artículo en existencia se le asigna un número de menos de ocho dígitos, por ejemplo, el 445, el sistema lo ajustará automáticamente por la derecha. Es decir que introducir 445 es lo mismo que introducir 00445 o 00000445.

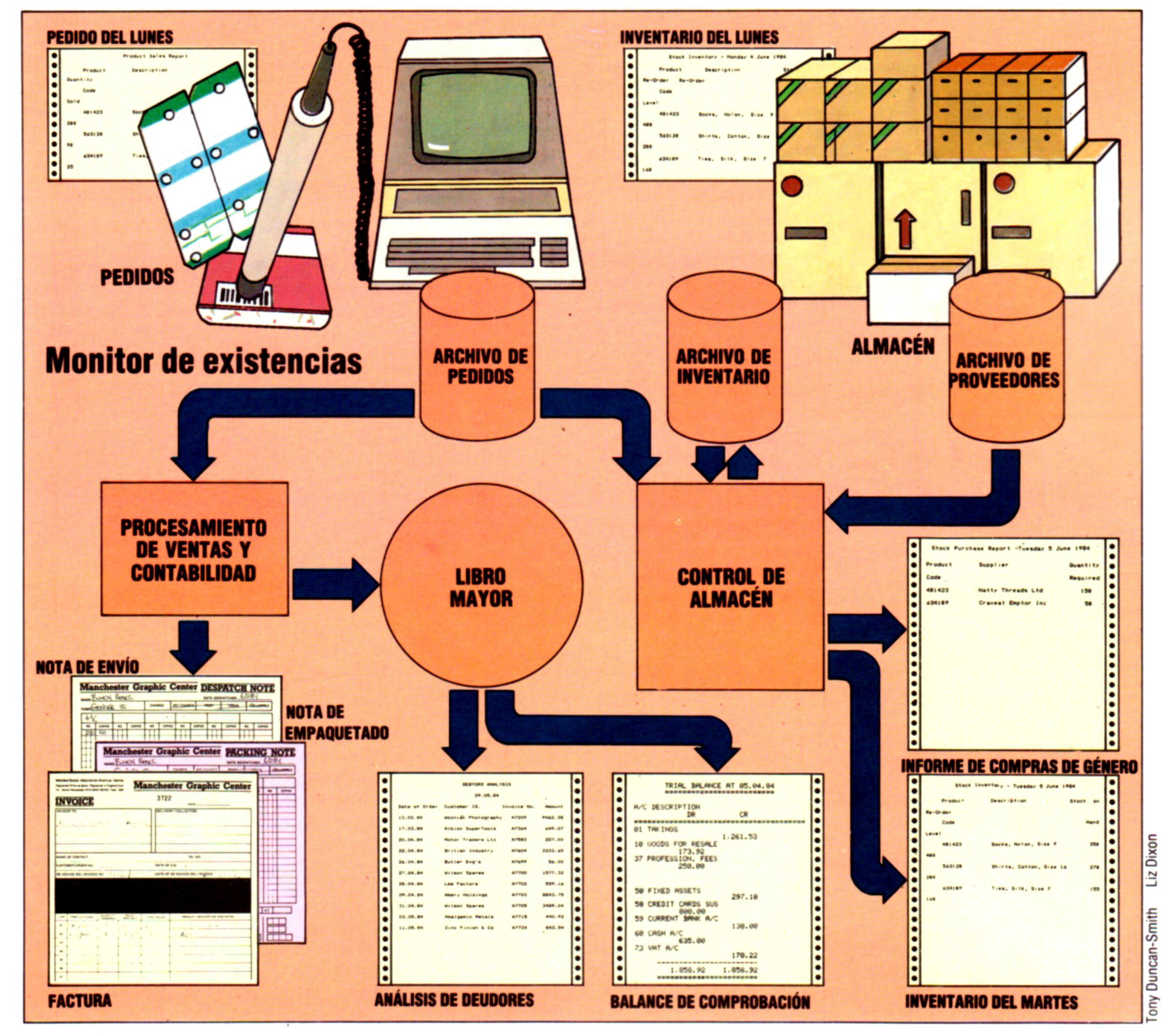
Esta cuestión de la alineación es importante. Por ejemplo, el *Stock control system* de ACT Pulsar, que opera en micros más poderosos como el IBM PC y el Sirius, permite a los usuarios elegir entre un sistema de codificación de ajuste por la derecha o por la izquierda. De lo cual resulta que esos sistemas de codificación de existencias son totalmente distintos e incompatibles.

El código del producto puede tener hasta 16 caracteres alfanuméricos. El sistema de ajuste por la derecha es un sistema de códigos numéricamente ordenados. El sistema de ajuste por la izquierda está dirigido a usuarios que cuentan con sistemas de codificación bastante más complejos y que incluyen alfanuméricos como el PX445/44. Permite tener códigos de diversas longitudes para distintos productos y resulta útil en los sistemas en los que el usuario quiere emplear el código de producto para identificar alguna característica del artículo en existencia, como puede ser el modelo, la talla o el color de unas prendas determinadas.

El paquete *Powerstock* de Omicron, que se utiliza en el Sirius, es un sistema más costoso destinado a usuarios cuyas necesidades son más complejas. Presenta un sistema de codificación aún más sutil y se define por grupos de existencias. Un grupo de existencias puede ser cualquier conjunto de registros de existencias relacionados por un proceso común o por necesidades de información. Lo que lo diferencia del sistema de codificación *Pulsar* de ajuste por la izquierda es que cada grupo de existencias se procesa por separado y a cada uno se le pueden asignar diversas reglas de procesamiento. Recuerde que cualquiera que sea el número de código asignado a una línea de productos en el *Pulsar*, todos los números de código se procesan de la misma manera.

En consecuencia, la estructura codificadora de los sistemas de control de existencias ha de ser lo bastante flexible para que los usuarios puedan identificar y subdividir los artículos en existencia. Los sistemas más simples que operan en los ordenadores personales baratos suelen ofrecen menos flexibilidad debido a las limitaciones, una vez más, de memoria y almacenamiento. El sistema de Dragon Data, por ejemplo, está diseñado para manejar un máximo de 350 artículos en existencia. El sistema *Powerstock* de Omicron no es restrictivo y su número máximo de artículos depende de la configuración del ordenador del usuario

Control de almacén
Los sistemas integrados de ventas y de control de existencias de las empresas con gran movimiento pueden beneficiarse enormemente con un sistema automatizado de punto de venta que permite mantener al día el inventario y los libros de contabilidad. Los datos de venta pueden incorporarse en un precinto a modo de una etiqueta Kimball o un código de barras pegado al producto. Éstos son captados por un lector óptico adosado a la caja registradora, que a su vez podría ser un microordenador, o transmitir los datos a un ordenador para su procesamiento



Tony Duncan-Smith Liz Dixon

Grados de precisión

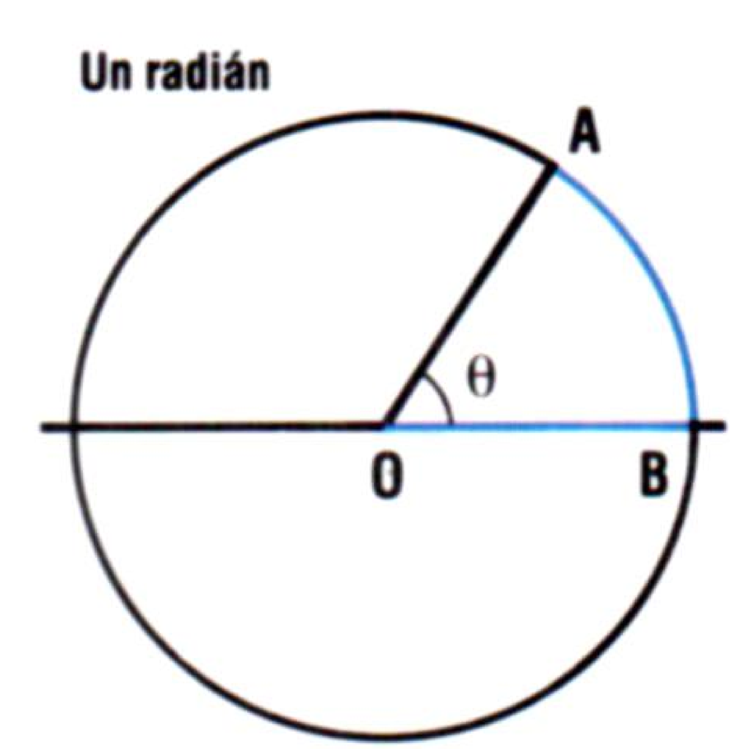
Esta vez estudiaremos qué tratamiento tienen las funciones seno y coseno en los programas BASIC y los métodos de comprobación de ambas para detectar posibles fuentes de error

Puesto que el BASIC cuenta con las funciones COS (coseno) y SIN (seno), será tarea fácil calcular la posición de un punto en una línea después de hacerlo rotar un cierto número de grados. El COS de θ nos dará un punto en el *eje x* (la abscisa *x*) y el SIN de θ dará el punto en el *eje y* (la ordenada *y*). No obstante, a la hora de emplear estas funciones conviene recordar que la mayoría de las funciones del BASIC operan en radianes y no en grados. Otra cuestión que se debe controlar es que los valores de θ dados a conocer pueden no ser fiables a medida que θ se aproxima a 0 o a 1. Pronto comprenderá la diferencia que hay entre grados y radianes.

Cuando se dibuja una porción de circunferencia (denominada *arco*) con longitud exactamente igual al radio, al ángulo central se le llama radián (véase la ilustración). Si además el radio del círculo lo tomamos como unidad de medida, esta porción de la circunferencia tendrá la longitud de una unidad. La fórmula para hallar la longitud de una circunferencia es 2πr, por lo que el recorrido de una vuelta completa medirá 2π radianes. Pero según una expresión más conocida, una revolución completa —el giro necesario para trazar una circunferencia— es de 360°. Por lo tanto, esos 360° equivalen a 2π radianes. Y así tenemos un modo sencillo de relacionar grados y radianes:

$360^\circ = 2\pi$ radianes
$180^\circ = \pi$ radianes
$90^\circ = \pi/2$ radianes
$1^\circ = \pi/180 = 0{,}0174$ radianes

Un radián



Cuando la longitud del radio, OB, se toma como unidad y el arco AB mide lo mismo que el radio, entonces al ángulo central θ se le llama *radián*. Muy brevemente: si $\overline{OB} = \widehat{BA}$ entonces $\widehat{AOB} = 1$ radián

Un programa BASIC que tuviera que hallar el coseno de un ángulo medido en grados, primero tendría que convertir la medida del ángulo de grados en radianes y luego utilizar la función COS. Pruebe con esto:

```
10 INPUT "DE ENTRADA ANGULO EN GRADOS";A
20 LET B# = A* 0,0174
30 LET C# = 2 COS (B#)
40 PRINT "EL COSENO DE  ";A; "  GRADOS ES  ";C#
50 END
```

El símbolo # indica que las variables del programa son de doble precisión (cuestión que abordaremos más adelante en este mismo capítulo). Una sencilla modificación de dicho programa utilizando la función seno dará entrada a todos los valores de θ de 0° a 360° y proporcionará el seno de dichos valores en forma de tabla. Si dichos valores se pasan al *eje y* de una gráfica (donde el *eje x* representa los valores de θ en radianes), aparecerá el gráfico de onda de seno conocido por los entusiastas de la alta fidelidad y los ingenieros electrónicos (véase el diagrama p. 635). Esta conocida curva no es más que la representación de los puntos de la intersección de la hipotenusa con la circunferencia unitaria sobre el *eje y* para todos los ángulos de rotación. En síntesis, es un modo alternativo de describir matemáticamente una circunferencia.

Unas pocas versiones de BASIC permiten que las funciones SIN y COS operen en grados o en radianes mediante la utilización de un "interruptor de software", pero la mayoría de éstas no cuentan con él. Si prefiere trabajar siempre en grados, es posible recurrir a una "función definida por el usuario" que le facilite las conversiones. Ésta es una de tantas posibles:

```
10 REM FUNCION DEFINIDA POR EL USUARIO
   PARA TRABAJAR EN GRADOS
20 DEF FNGSIN (G#) = SIN(G#*0.017453293)
30 INPUT "DE ENTRADA AL ANGULO EN
   GRADOS";G#
40 PRINT "EL SENO DE  ";G#;"  GRADOS ES";
   FNGSIN(G#)
50 END
```

La línea 20 define una función denominada GSIN (que significa grados-seno), que utiliza como único parámetro la variable de doble precisión G#. La mitad derecha de la definición sólo muestra cómo debe obtenerse el valor que ha de ser calculado por la función (el seno de un ángulo en grados). Para echar mano de una función definida por el usuario, basta con que utilice como de costumbre el nombre de la función (poniendo entre paréntesis el valor a operar). No obstante, note que la línea que contiene la definición tiene que ser ejecutada antes de poder llamar a la función.

Uno de los problemas que surgen al usar la función seno en BASIC es que no todas las versiones la manejan correctamente cuando se aproxima a 0 el valor de θ. Es obvio que, a medida que θ se aproximará a 0, el valor de SIN θ también se aproximará a 0 (véase p. 634). En síntesis, a medida que el ángulo se aproxima a 0, el arco de la circunferencia que define θ también se acerca a 0, y el punto en que la hipotenusa corta el círculo se acerca a 0 en el *eje y*. Por desgracia, la precisión del BASIC es limitada. Dicho de otro modo, este lenguaje sólo puede manejar valores grandes hasta cierto límite y valores pequeños hasta cierto valor también. Si θ es muy pequeño (p. ej., 1.0E-36, es decir, 1 × 10 elevado a la menos 36), es posible que el BASIC no pueda resolverlo y simplemente devuelva un valor de 0 para el seno de dichos números. Antes de utilizar la función seno, intente someter a prueba su BASIC poniendo en práctica el programita que detallamos a continuación:

Errores de redondeo por aproximación a cero

```
1 REM PRUEBA DE ERRORES PARA LA FUNCION SIN PROXIMA A CERO
10 LEXT X = 10/6
20 PRINT "ITERACION"," VAL DE X"," VAL DE SIN(X)"
30 FOR I = I TO 40
40 LET X = X / 10
50 PRINT I,X,SIN(X)
60 NEXT I
70 END
```

ITERACIÓN	VAL DE X	VAL DE SIN(X)
1	.166667	.165896
2	.0166667	.0166659
3	1.66667E-03	1.66667E-03
4	1.66667E-04	1.66667E-04
5	1.66667E-05	1.66667E-05
6	1.66667E-06	1.66667E-06
7	1.66667E-07	1.66667E-07
8	1.66667E-08	1.66667E-08
9	1.66667E-09	1.66667E-09
10	1.66667E-10	1.66667E-10
11	1.66667E-11	1.66667E-11
12	1.66667E-12	1.66667E-12
13	1.66667E-13	1.66667E-13
14	1.66667E-14	1.66667E-14
15	1.66667E-15	1.66667E-15
16	1.66667E-16	1.66667E-16
17	1.66667E-17	1.66667E-17
18	1.66667E-18	1.66667E-18
19	1.66667E-19	1.66667E-19
20	1.66667E-20	1.66667E-20
21	1.66667E-21	1.66667E-21
22	1.66667E-22	1.66667E-22
23	1.66667E-23	1.66667E-23
24	1.66667E-24	1.66667E-24
25	1.66667E-25	1.66667E-25
26	1.66667E-26	1.66667E-26
27	1.66667E-27	1.66667E-27
28	1.66667E-28	1.66667E-28
29	1.66667E-29	1.66667E-29
30	1.66667E-30	1.66667E-30
31	1.66667E-31	1.66667E-31
32	1.66667E-32	1.66667E-32
33	1.66667E-33	1.66667E-33
34	1.66667E-34	1.66667E-34
35	1.66667E-35	1.66667E-35
36	1.66667E-36	1.66667E-36
37	1.66667E-37	1.66667E-37
38	1.66667E-38	1.66667E-38
39	0	0
40	0	0

Damos también una ejecución de este programa utilizando MBASIC Microsoft. Este intérprete concreto de BASIC maneja bastante bien el SIN de pequeñas cifras y no crea problemas hasta que el valor de θ es menor que 10E-38 (una coma decimal seguida de 37 ceros).

Este programa depende de un margen dinámico adecuado en el manejo que el BASIC haga de las operaciones aritméticas de coma flotante. Conviene recordar que antes de que usted pueda utilizar con seguridad cualquier operación matemática del BASIC tendrá que saber cuál es el margen de números que puede manejar con exactitud.

Recuerde que el nombre de variable, como por ejemplo X o TREND, será automáticamente de *precisión simple* (es decir, capaz sólo de almacenar hasta siete dígitos). Por otra parte, las variables pueden especificarse como de, o pasarse a, precisión simple añadiendo un signo de exclamación, como en X! o TREND!. Las variables de *doble precisión* (que pueden almacenar hasta 17 dígitos) se especifican añadiendo el símbolo #, como en X# o TREND#. Las variables que sólo pueden almacenar números enteros se especifican en muchas versiones mediante el agregado del símbolo de porcentaje, como en X% o TREND%.

Concluimos este artículo con un breve programa que le permite comprobar cuántos dígitos pueden almacenarse en una variable de su versión de BASIC, así como una salida impresa del programa cuando funciona utilizando BASIC Microsoft. Hay dos versiones, una para probar cifras pequeñas y otra para grandes. La salida impresa de las cifras pequeñas muestra que a medida que los números se vuelven muy pequeños (menos que 3,3 × 10E-38), el BASIC redondea los números a cero. En las cifras grandes (mayores de 3,3 × 10E37), se produce un desbordamiento y los resultados son poco fiables. Es posible que si necesita operar con cifras muy grandes o muy pequeñas, tenga que preparar rutinas aritméticas específicas para superar dichas limitaciones.

Los números pequeños en BASIC

```
1 REM PRUEBAS CON NUMS. PEQUEÑOS EN BASIC
10 LET X# = .00003333333333#
20 PRINT "ITERACION"," DOBL PREC"," ","SIMPL PREC"
30 PRINT
40 FOR I = 1 TO 40
50 LET X# = X# / 10
60 LET X! = X#
70 PRINT I,X#,X!
80 NEXT I
90 END
```

ITERACIÓN	DOBL PREC		SIMPL PREC
1	.000003333333333		3.33333E-06
2	.0000003333333333		3.33333E-07
3	3.333333333D-08		3.33333E-08
4	3.333333333D-09		3.33333E-09
5	3.333333333D-10		3.33333E-10
6	3.333333333D-11		3.33333E-11
7	3.333333333D-12		3.33333E-12
8	3.333333333D-13		3.33333E-13
9	3.333333333D-14		3.33333E-14
10	3.333333333D-15		3.33333E-15
11	3.333333333D-16		3.33333E-16
12	3.333333333D-17		3.33333E-17
13	3.333333333D-18		3.33333E-18
14	3.333333333D-19		3.33333E-19
15	3.333333333D-20		3.33333E-20
16	3.333333333D-21		3.33333E-21
17	3.333333333D-22		3.33333E-22
18	3.333333333D-23		3.33333E-23
19	3.333333333D-24		3.33333E-24
20	3.333333333D-25		3.33333E-25
21	3.333333333D-26		3.33333E-26
22	3.333333333D-27		3.33333E-27
23	3.333333333D-28		3.33333E-28
24	3.333333333D-29		3.33333E-29
25	3.333333333D-30		3.33333E-30
26	3.333333333D-31		3.33333E-31
27	3.333333333D-32		3.33333E-32
28	3.333333333D-33		3.33333E-33
29	3.333333333D-34		3.33333E-34
30	3.333333333D-35		3.33333E-35
31	3.333333333D-36		3.33333E-36
32	3.333333333D-37		3.33333E-37
33	3.333333333D-38		3.33333E-38
34	0	0	0
35	0	0	0
36	0	0	0
37	0	0	0
38	0	0	0
39	0	0	0
40	0	0	0

Los números grandes en BASIC

```
1 REM PRUEBAS CON NUMS. GRANDES EN BASIC
10 LET X# = 3.333333333333334#
20 PRINT "ITERACION"," DOBL PREC", ","," SIMPL PREC"
30 PRINT
40 FOR I = 1 TO 40
50 LET X# = X# * 10
60 LET X! = X#
70 PRINT I,X#,X!
80 NEXT I
90 END
```

ITERACION	DOBL PREC	SIMPL PREC
1	33.33333333333334	33.3333
2	333.3333333333334	333.333
3	3333.333333333334	3333.33
4	33333.33333333334	33333.3
5	333333.3333333334	333333
6	3333333.333333334	3.33333E + 06
7	33333333.33333334	3.33333E + 07
8	333333333.3333334	3.33333E + 08
9	3333333333.333334	3.33333E + 09
10	33333333333.33334	3.33333E + 10
11	333333333333.3334	3.33333E + 11
12	3333333333333.334	3.33333E + 12
13	33333333333333.34	3.33333E + 13
14	333333333333333.4	3.33333E + 14
15	3333333333333334	3.33333E + 15
16	3.333333333333334D + 16	3.33333E + 16
17	3.333333333333334D + 17	3.33333E + 17
18	3.333333333333334D + 18	3.33333E + 18
19	3.333333333333334D + 19	3.33333E + 19
20	3.333333333333334D + 20	3.33333E + 20
21	3.333333333333334D + 21	3.33333E + 21
22	3.333333333333334D + 22	3.33333E + 22
23	3.333333333333334D + 23	3.33333E + 23
24	3.333333333333334D + 24	3.33333E + 24
25	3.333333333333334D + 25	3.33333E + 25
26	3.333333333333334D + 26	3.33333E + 26
27	3.333333333333334D + 27	3.33333E + 27
28	3.333333333333334D + 28	3.33333E + 28
29	3.333333333333334D + 29	3.33333E + 29
30	3.333333333333334D + 30	3.33333E + 30
31	3.333333333333334D + 31	3.33333E + 31
32	3.333333333333334D + 32	3.33333E + 32
33	3.333333333333334D + 33	3.33333E + 33
34	3.333333333333334D + 34	3.33333E + 34
35	3.333333333333334D + 35	3.33333E + 35
36	3.333333333333334D + 36	3.33333E + 36
37	3.333333333333334D + 37	3.33333E + 37
38	1.701411834604693D + 38	1.70141E + 38
39	1.701411834604693D + 38	1.70141E + 38
40	1.701411834604693D + 38	1.70141E + 38

Banderín de salida

Daremos una atenta mirada a la instrucción suma (ADD) y al "registro indicador de estado" (PSR), cuyo indicador de arrastre es fundamental en la suma

La instrucción suma en el lenguaje assembly tanto del Z80 como del 6502 es ADC (abreviatura de "ADd with Carry": suma con arrastre), una mnemotecnia de gran importancia para la programación en dicho lenguaje. El concepto de un bit "de arrastre" tiene especial significado. Consideremos la suma de dos números hexadecimales en el acumulador:

	A7	=	10100111
+	3E	=	+ 00111110
	E5	=	11100101

Puesto que el acumulador es un registro de ocho bits, tanto los números a sumar como la suma propiamente dicha no podrán superar el intervalo de 0000 0000 a 1111 1111 en binario, o sea de $00 a $FF en hexa (como aparecen aquí), pues de lo contrario no encajarán en el acumulador. En consecuencia, ¿ello significa que estamos limitados a adiciones en las que la suma es inferior a $100 (en decimal, 256)? Veamos otra adición en el acumulador, una que viole dicha restricción:

	FF	=	11111111
+	FF	=	+ 11111111
	1FE	=	111111110

Se trata de la adición de dos números de un solo byte lo más grandes posible. La suma no parece viable, pues requiere un acumulador de nueve bits y el que está a nuestra disposición es de ocho. La solución de la dificultad queda sugerida en el planteamiento del problema: sólo necesitamos un bit adicional en el acumulador para contener el número más grande que puede generar la suma de cifras de un solo byte. El bit adicional sólo hace falta en la suma, no en los operandos de la adición, y en caso de que haya un "arrastre" del bit más significativo del acumulador.

Registro indicador de estado

Así pues, ese bit adicional se conoce como "bit de arrastre" (*carry bit*) y está localizado en un registro compuesto de ocho bits asociado con el acumulador y conocido como *registro indicador de estado* (*Processor Status Register*: PSR). Este importante registro está conectado con el acumulador y la ALU de modo tal que los bits individuales del PSR se activan o se desactivan después de cualquier operación en el acumulador, según sean los resultados de dicha operación. El contenido del registro indicador de estado puede considerarse como un número simple, pero con frecuencia resulta más ilustrativo tratarlo como una disposición de ocho indicadores simples, que reciben el nombre de *flags* o *banderas*, cuyo estado individual muestra los efectos específicos de la última operación (un flag es una variable cuyo valor indica la verdad de alguna condición, por lo que no se toma como valor absoluto. Una variable de flag sólo acostumbra presentar dos estados o condiciones: sí o no, activado o desactivado, 0 o 1).

Cuando en el acumulador se realiza cualquier operación que produce el arrastre del octavo bit, el flag de arrastre del PSR se posicionará automáticamente en 1; toda operación que no provoque un arrastre restablecerá (o posicionará en 0) el flag de arrastre. Esto sólo se aplica a operaciones que podrían provocar un arrastre. Algunas operaciones como cargar al o almacenar desde el acumulador no afectan al flag de arrastre. A partir de este punto, cada vez que abordemos una nueva instrucción en lenguaje assembly, nos interesará saber cuáles son los bits del PSR (o registro de flag) afectados. Lógicamente, tendremos que saber más cosas sobre los otros flags del PSR, pero concluyamos antes nuestro análisis del flag de arrastre.

En la mayoría de los casos, cuando sumemos dos números de un solo byte, no sabremos por adelantado cuáles serán, por lo que tendremos que estar dispuestos a que el resultado de dicha suma supere $FF; esto suele suponer reservar dos bytes de RAM para contener el resultado de la suma. Volvamos una vez más a los ejemplos anteriores:

	Hexadecimal		Flag de arrastre	Binario
	A7	=		10100111
+	3E	=		+ 00111110
	00E5	=	0	11100101
			No arrastre	
	FF	=		11111111
+	FF	=		+ 11111111
	01FE	=	1	111111110
			Arrastre	

En ambas sumas, el resultado de la adición está representado por un número de dos bytes. En el primer caso, el flag de arrastre se restablece en cero, porque no hay arrastre del octavo bit en la suma (el resultado de dos bytes es $00E5, del que el byte *hi* —o byte alto— es $00). Pero como en el segundo caso hay un arrastre del octavo bit, el flag de arrastre se enciende y el byte *hi* del resultado es $01.

Por lo tanto, para comprobar que obtenemos el resultado correcto de una adición, debemos almacenar el contenido del acumulador en el byte *lo* (o byte bajo) de la posición de dos bytes y a continua-

ción almacenar el flag de arrastre como byte *hi* de dicha posición. No existe una instrucción exclusiva para almacenar el flag de arrastre, pero el código operativo ADC fue formulado pensando en dicha operación: ADC significa realmente "sumar el operando de la instrucción al contenido actual del flag de arrastre y añadir luego ese resultado al contenido del acumulador". Así, la adición es un proceso en dos etapas; en la primera se utiliza el estado actual del flag de arrastre, mientras que en la segunda se actualiza el estado de dicho flag de arrastre.

Todo ello significa que antes de iniciar la suma debemos considerar el estado actual del flag de arrastre, ya que será sumado al resultado de la suma propiamente dicha: de ahí que se disponga de CLC y AND A, las dos instrucciones no explicadas en capítulos anteriores. La primera, una instrucción del 6502, significa "desactivar el flag de arrastre" (*CLear the Carry flag*), y es eso exactamente lo que hace. La versión del Z80, AND A, significa "activar un AND (y) lógico entre el acumulador y él mismo". Aunque no está diseñada exclusivamente para restaurar el flag de arrastre, ése es el efecto que produce y no afecta nada más, por lo que a menudo se la utiliza como el equivalente de la CLC del 6502 para el Z80.

Por lo tanto, luego de desactivar el flag de arrastre y tras realizar la suma, a continuación debemos almacenar su contenido. Ello se consigue añadiendo el valor inmediato $00 al byte *hi* del resultado. Esto no afectará al byte si el flag de arrastre está desactivado, pero le sumará 1 si se halla activado.

Todo lo que hemos dicho en este capítulo no es sino el primer método de la aritmética de un solo byte. En resumen, se deberá:

1) Desactivar el flag de arrastre
2) Cargar el acumulador con un número
3) Sumarle el segundo número
4) Almacenar el contenido del acumulador en el byte *lo* de una posición
5) Cargar el acumulador con el contenido del byte *hi*
6) Sumarle el valor inmediato $00
7) Almacenar el contenido del acumulador en el byte *hi*

Cuando este procedimiento se convierte a lenguaje assembly, obtenemos:

COMÚN PARA AMBOS PROCESADORES		
Etiqueta	**Directriz**	**Operando**
BYTE1	EQU	$FF
BYTE2	EQU	$FF
LOBYTE	EQU	$A000
HIBYTE	EQU	$A001
	ORG	$A020

Z80		6502	
Opcode	**Operando**	**Opcode**	**Operando**
LD	A,$00	LDA	#$00
LD	(HIBYTE),A	STA	HIBYTE
AND	A	CLC	
LD	A,BYTE1	LDA	#BYTE1
ADC	A,BYTE2	ADC	#BYTE2
LD	(LOBYTE),A	STA	LOBYTE
LD	A,(HIBYTE)	LDA	HIBYTE
ADC	A,$00	ADC	#$00
LD	(HIBYTE),A	STA	HIBYTE
RET		RTS	

Recuerde que los valores dados a LOBYTE, HIBYTE y ORG sólo son a modo de ejemplo y que usted ha de escoger valores adecuados para la máquina que utiliza. Note que las dos primeras instrucciones del programa cargan $00 en el HIBYTE a fin de que se vea corrompida por datos aleatorios. No tenemos que desactivar LOBYTE del mismo modo porque su contenido de partida está sobreescrito en el byte *lo* del resultado.

Conviene resaltar una vez más las diferencias de enfoque entre el lenguaje assembly del Z80 y del 6502, como ya se ha visto en el ejemplo. El código del 6502 se interpreta fácilmente en cuanto uno se acostumbra a él; la mnemotecnia propiamente dicha y el uso de "#" para señalar datos inmediatos clarifica el significado de cada instrucción. La versión del Z80 es menos directa pues la mnemotecnia LD se utiliza para todas las transferencias de datos, ya sea que entren o salgan del acumulador. Además, no existe el símbolo "#" que señale datos inmediatos y sólo se indica mediante la ausencia de paréntesis en el operando. De este modo, LD A,BYTE1 significa "cargar el acumulador con los datos inmediatos BYTE1"; mientras que LD A,(HIBYTE) significa "cargar el acumulador desde la dirección HIBYTE". En la lista completa de lenguaje assembly no hay ambigüedad respecto del significado de dichas instrucciones ya que el valor hexadecimal del código operativo identifica exclusivamente la instrucción. Esto parecería dar por sentada la cuestión: el código operativo podría ser exclusivo, pero si hay diversas opciones de códigos operativos exclusivos, ¿cómo hace el ensamblador (o la persona que realiza el ensamblaje) para elegir entre ellos? La respuesta reside en la modalidad de direccionamiento, que será el tema que nos ocupará en el próximo capítulo.

Por último, debemos mencionar que el registro de estado del proceso contiene otros flags además del de arrastre, cuestión que ahora analizaremos fugazmente y sobre la que volveremos más adelante con mayor detalle:

PSR Z80	S	Z		H		P/V	N	C
Bit número (MSB → LSB)	7	6	5	4	3	2	1	0
PSR 6502	S	V		B	D	I	Z	C

BIT PSR	Z80
7	(S) = SIGNO
6	(Z) = CERO
5	no usado
4	(H) = MEDIO ARRASTRE
3	no usado
2	(P/V) = PAR./OVERFLOW
1	(N) = RESTA
0	(C) = ARRASTRE

6502	BIT PSR
(S) = SIGNO	7
(V) = OVERFLOW	6
no usado	5
(B) = RUPTURA	4
(D) = MODO BCD	3
(I) = INTERRUPCION	2
(Z) = CERO	1
(C) = ARRASTRE	0

Para nuestro objetivo, en este momento los flags importantes son el de arrastre, el de signo y el de cero. Hemos visto que después de una adición la bandera de arrastre contiene el valor del arrastre del octavo bit del acumulador. La bandera de signo siempre es una copia del octavo bit (bit 7) del acumulador y la bandera de cero se posiciona en 1 si el contenido del acumulador es cero y se reacomoda en 0 si el contenido no es cero.

Soluciones al ejercicio de assembly de la p. 638

1) Los programas ensamblados se ofrecen en el cuadro de la derecha.
Note que los símbolos BYTE1 y BYTE2 se utilizan como datos simbólicos inmediatos y también como direcciones simbólicas. Sin embargo, cuando se utilizan como direcciones, es necesario ensamblarlos en forma de dos bytes.

2) La instrucción "regreso de una subrutina" está ausente del final de ambos programas. En la versión del 6502, el código completo tendría que incluir la siguiente línea:

```
A00D          60          RTS
```

y la versión del Z80 necesita esta línea:

```
A00D          C9          RET
```

3) El valor $45 se carga en el registro de acumulador como dato inmediato y luego se le suma $45, de modo que el acumulador contiene el valor $8A (o sea, 45 + 45 = 8A en hexa). Este total acumulado se almacena después en la RAM, dirección $0045. El valor $38 es ahora añadido al acumulador como dato inmediato, por lo que el acumulador pasa a contener el valor $C2 ($8A + $38 = $C2). Finalmente, este total se almacena en la RAM en la posición $0038.

4) "Datos inmediatos" son aquellos que realmente se almacenan en la instrucción. En las instrucciones que vimos en los programas de ejercicios (como LDA #$9C y LD A,$E4), los valores $9C y $E4 son los datos a cargar en el acumulador. Están almacenados en las instrucciones de las que son operandos e implican el contenido del byte inmediatamente posterior al opcode. Si no se dispone de datos, debe almacenarse en alguna otra parte de la memoria y hay que referirse a él por su dirección más que por su valor.

5) Como el valor del BYTE1 es $45, si se escribe como dirección será la posición de memoria $0045. Sin lugar a dudas, esa dirección está en la página cero de la memoria.

Ejercicio

Tal vez queramos examinar el contenido del registro indicador de estado (PSR) y nos convenga mostrar este número como byte binario y no hexadecimal. Aquí tiene la versión de una "subrutina de conversión de decimal a binario" del Spectrum. En este ejercicio se le pide que lo incorpore en el programa monitor de la pág. 598.

```
7000 REM******S/R BYTE BINARIO*******
7001 REM*CONVIERTE UN NUMERO N (256)*
7002 REM*EN BINARIO DE 8 CARACTERES*
7003 REM*REPRESENTACION EN B$*
7010 B$ = ""
7020 FOR D = 8 TO 1 STEP-1
7030 LET N1 = INT(N/2)
7040 LET R = N-2*N1
7050 LET B$ = STR$(R) + B$
7060 LET N = N1
7070 NEXT D
7080 RETURN
```

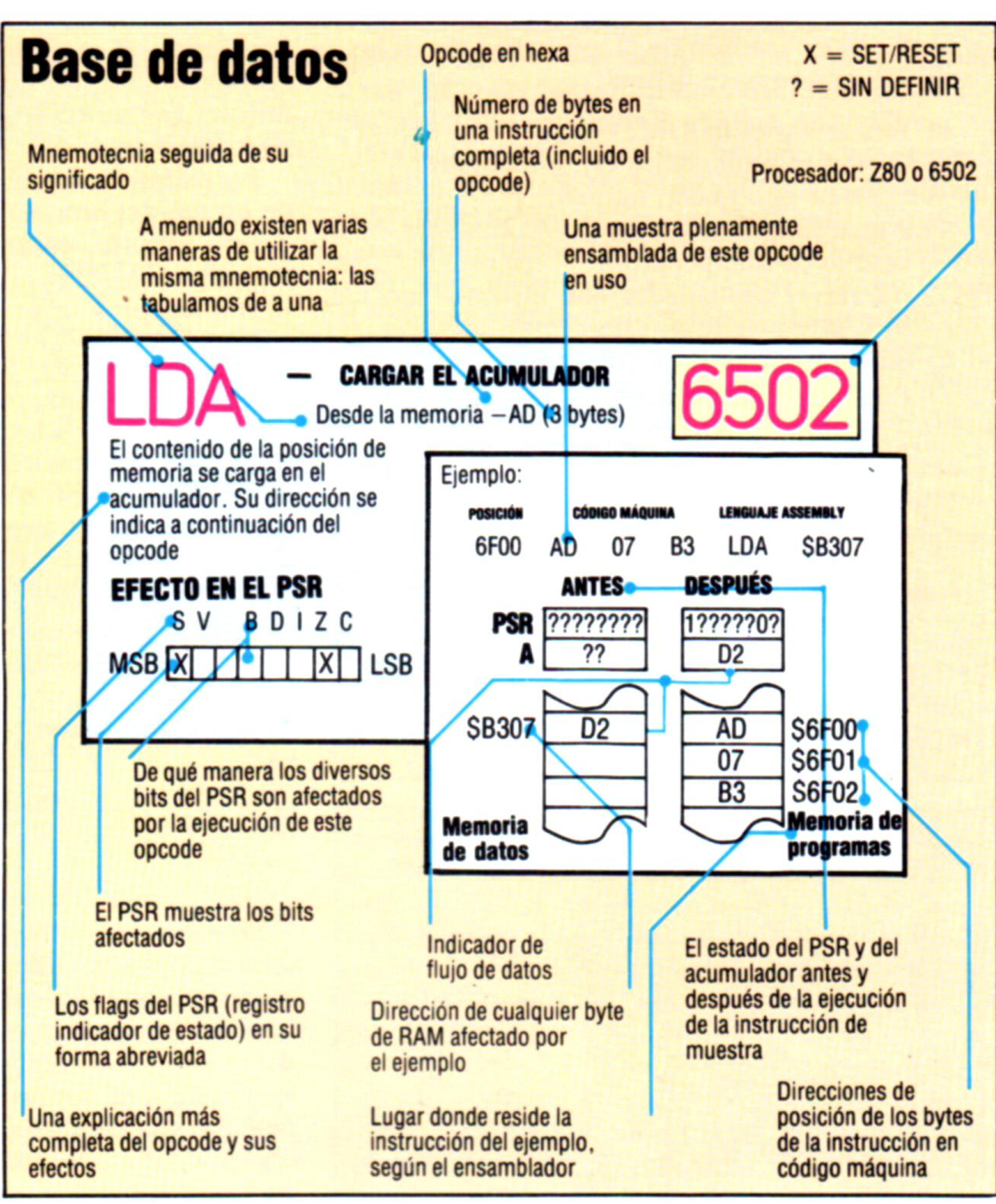
Complementos al BASIC

En el Commodore 64, cambie la línea 7050 de la subrutina por:

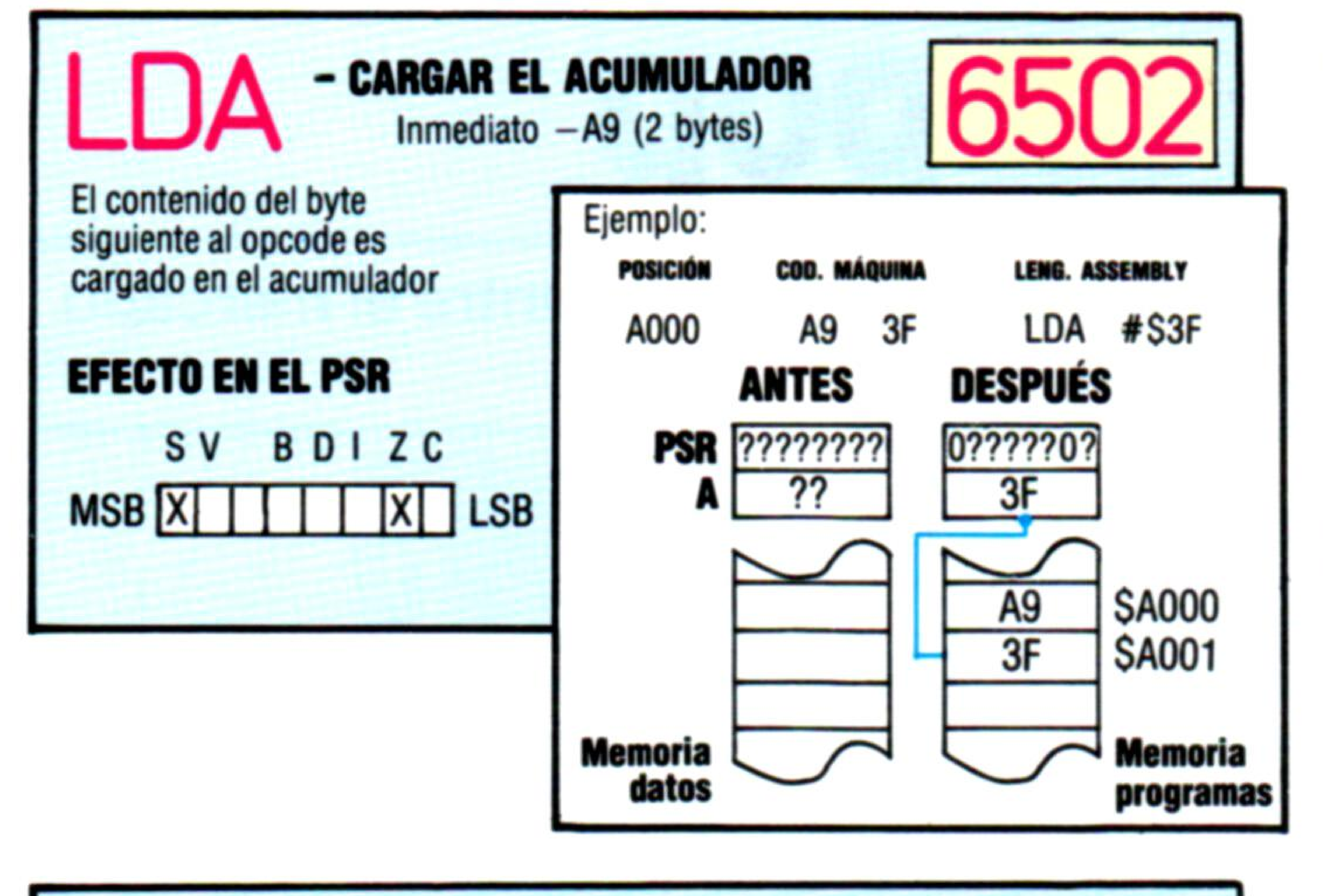
```
7050 B$ = MID$ (STR$(R),2) + B$
```

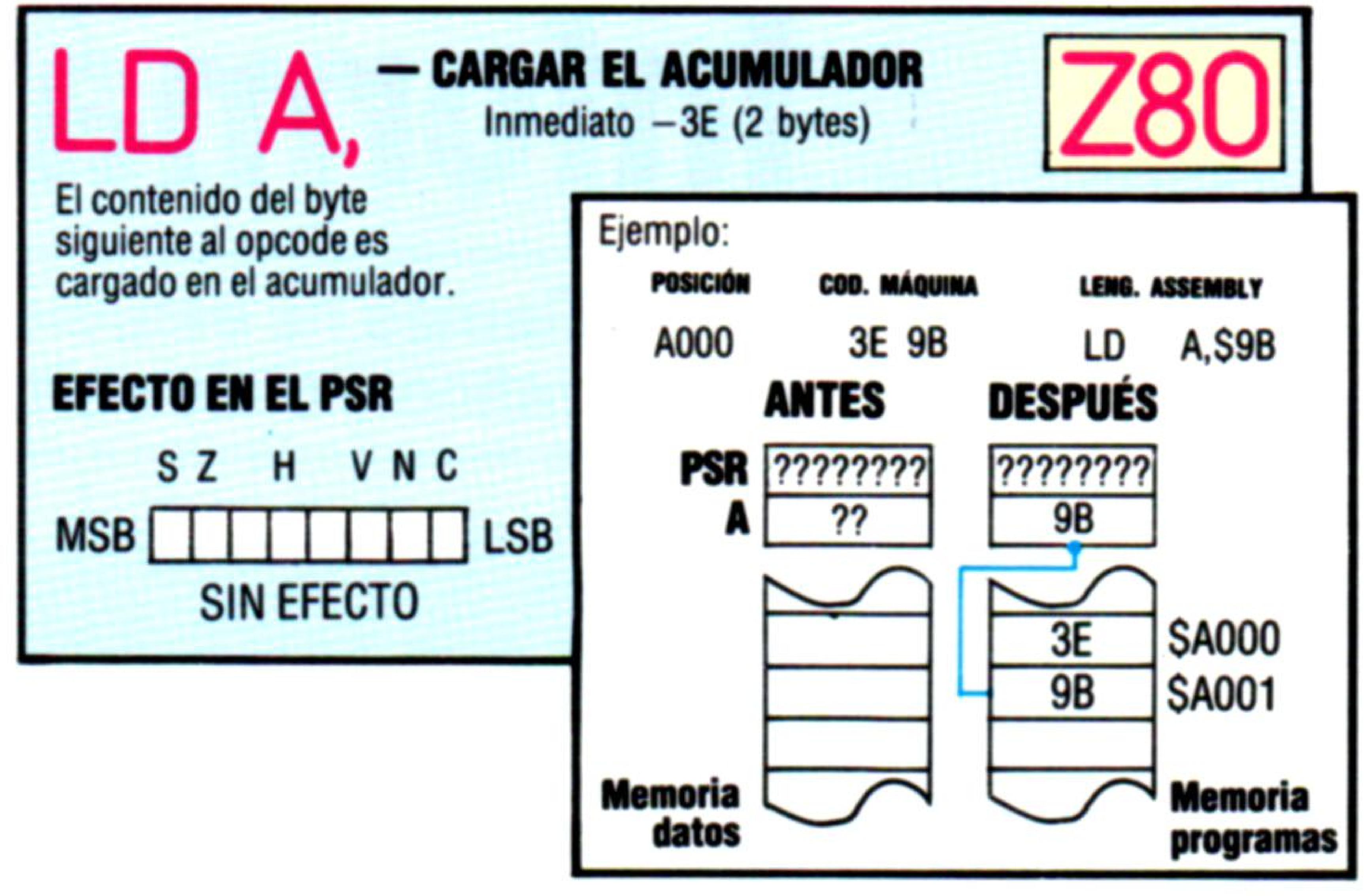
Dirección de posición	Lenguaje máquina	Lenguaje Assembly		
6502				
0000			EQU	$A000
0000		BYTE1	EQU	$45
0000		BYTE2	EQU	$38
0000			ORG	
A000	A9 45		LDA	#BYTE1
A002	18		CLC	
A003	69 45		ADC	#BYTE1
A005	8D 45 00		STA	BYTE1
A008	69 38		ADC	#BYTE2
A00A	8D 38 00		STA	BYTE2
Z80				
0000		START	EQU	$A000
0000		BYTE1	EQU	$45
0000		BYTE2	EQU	$38
0000			ORG	
A000	3E 45		LD	A,BYTE1
A002	A7		AND	A
A003	CE 45		ADC	A,BYTE1
A005	32 45 00		LD	(BYTE1),A
A008	CE 38		ADC	A,BYTE2
A00A	32 38 00		LD	(BYTE2),A

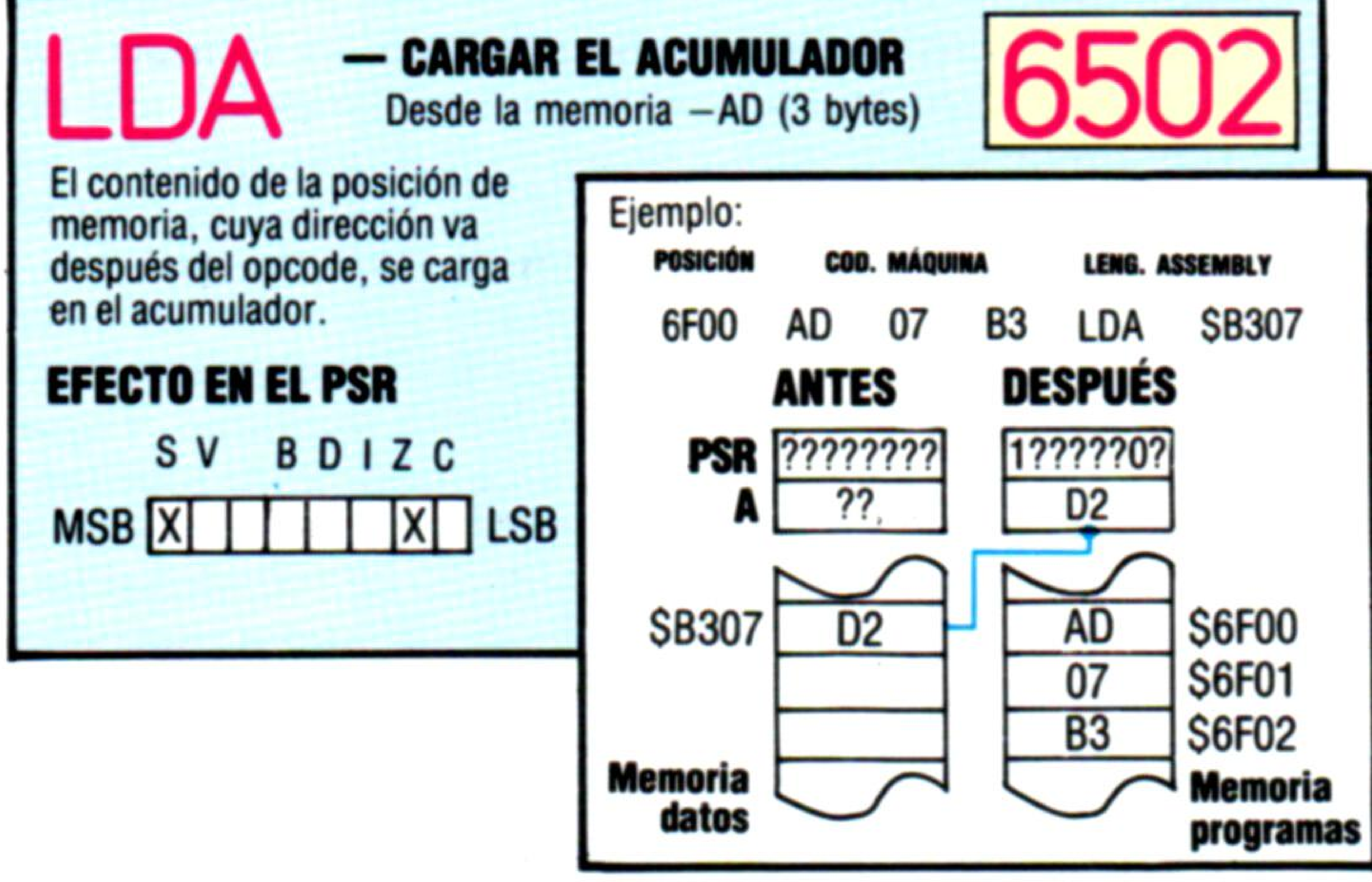
Base de datos

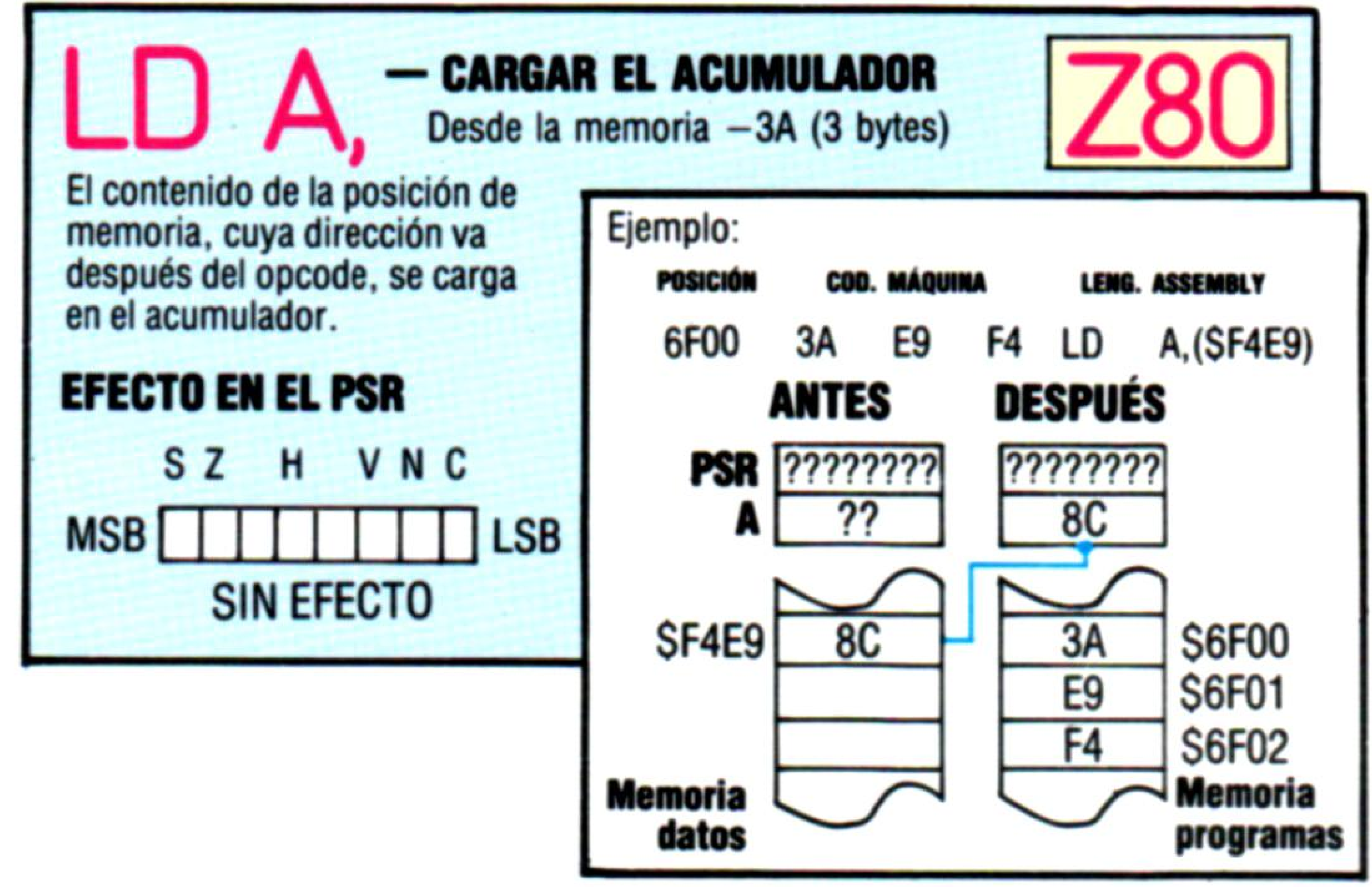


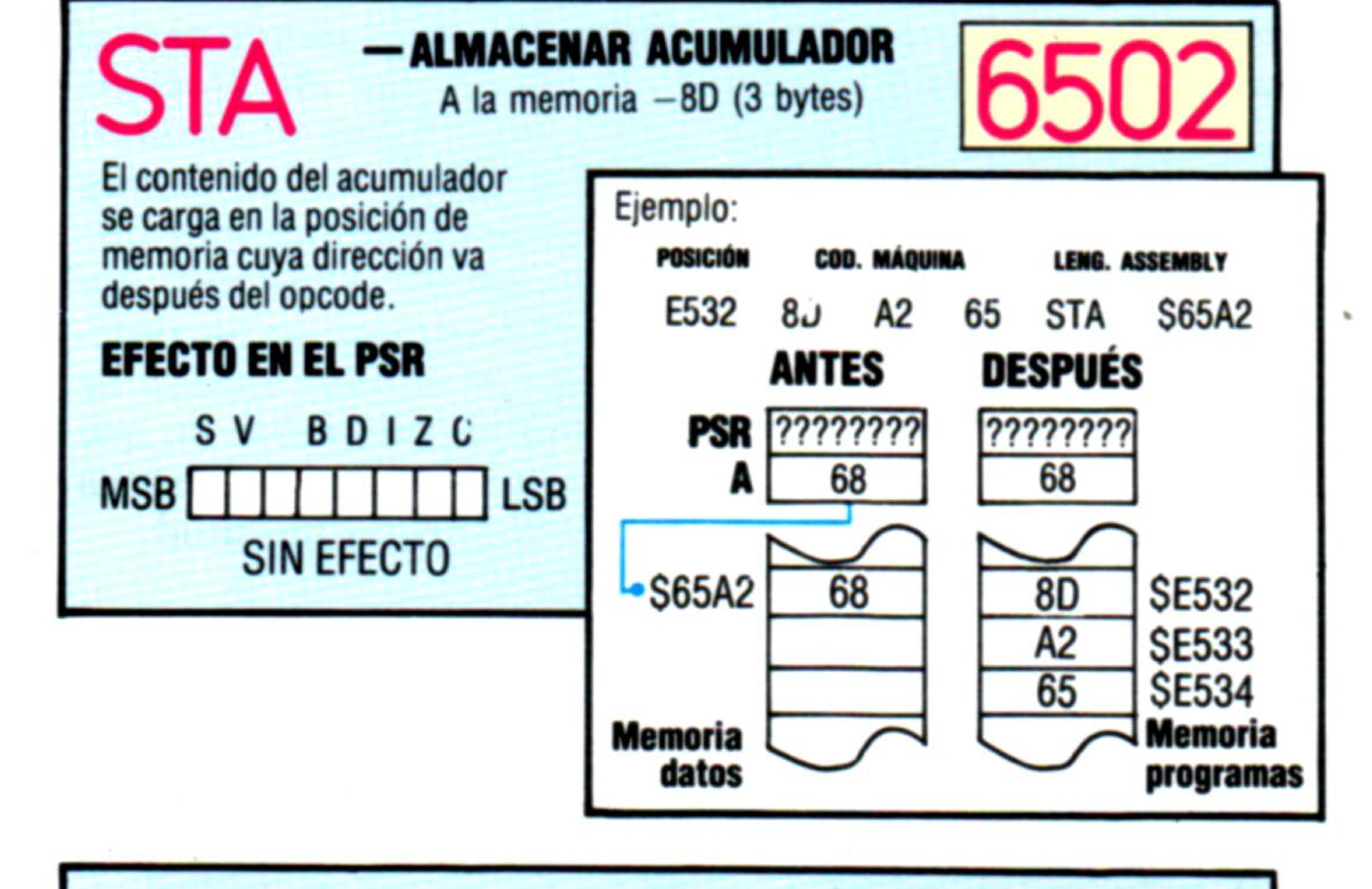


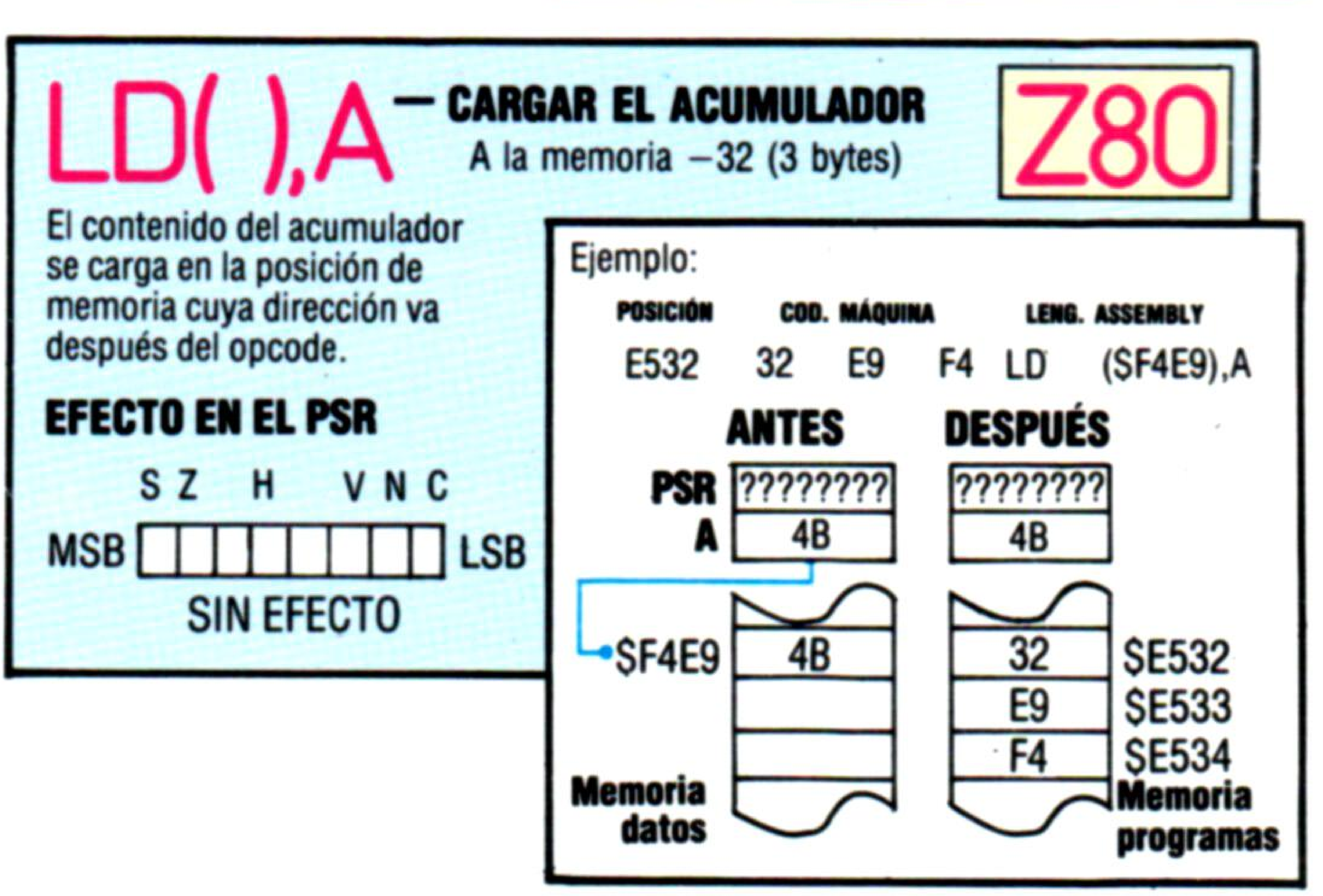

LDA
- CARGAR EL ACUMULADOR
Inmediato –A9 (2 bytes)
6502
El contenido del byte siguiente al opcode es cargado en el acumulador
EFECTO EN EL PSR
S V B D I Z C
MSB X X LSB
Ejemplo:
POSICIÓN
COD. MÁQUINA
LENG. ASSEMBLY
A000
A9 3F
LDA #$3F
ANTES
DESPUÉS
PSR
A
????????
??
0?????0?
3F
A9
3F
$A000
$A001
Memoria datos
Memoria programas

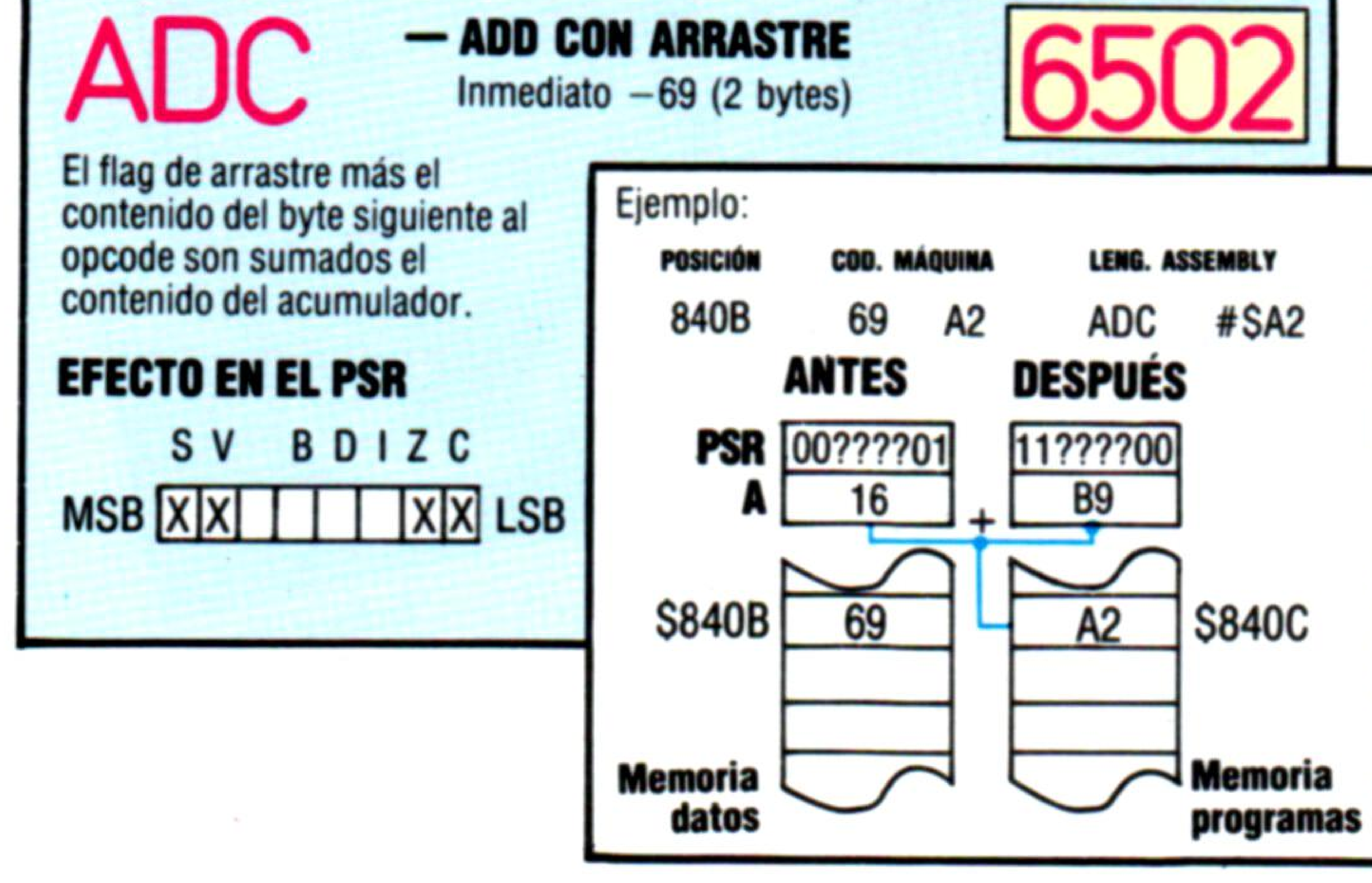

LD A,
– CARGAR EL ACUMULADOR
Inmediato –3E (2 bytes)
Z80
El contenido del byte siguiente al opcode es cargado en el acumulador.
EFECTO EN EL PSR
S Z H V N C
MSB LSB
SIN EFECTO
Ejemplo:
POSICIÓN
COD. MÁQUINA
LENG. ASSEMBLY
A000
3E 9B
LD A,$9B
ANTES
DESPUÉS
PSR
A
????????
??
????????
9B
3E
9B
$A000
$A001
Memoria datos
Memoria programas

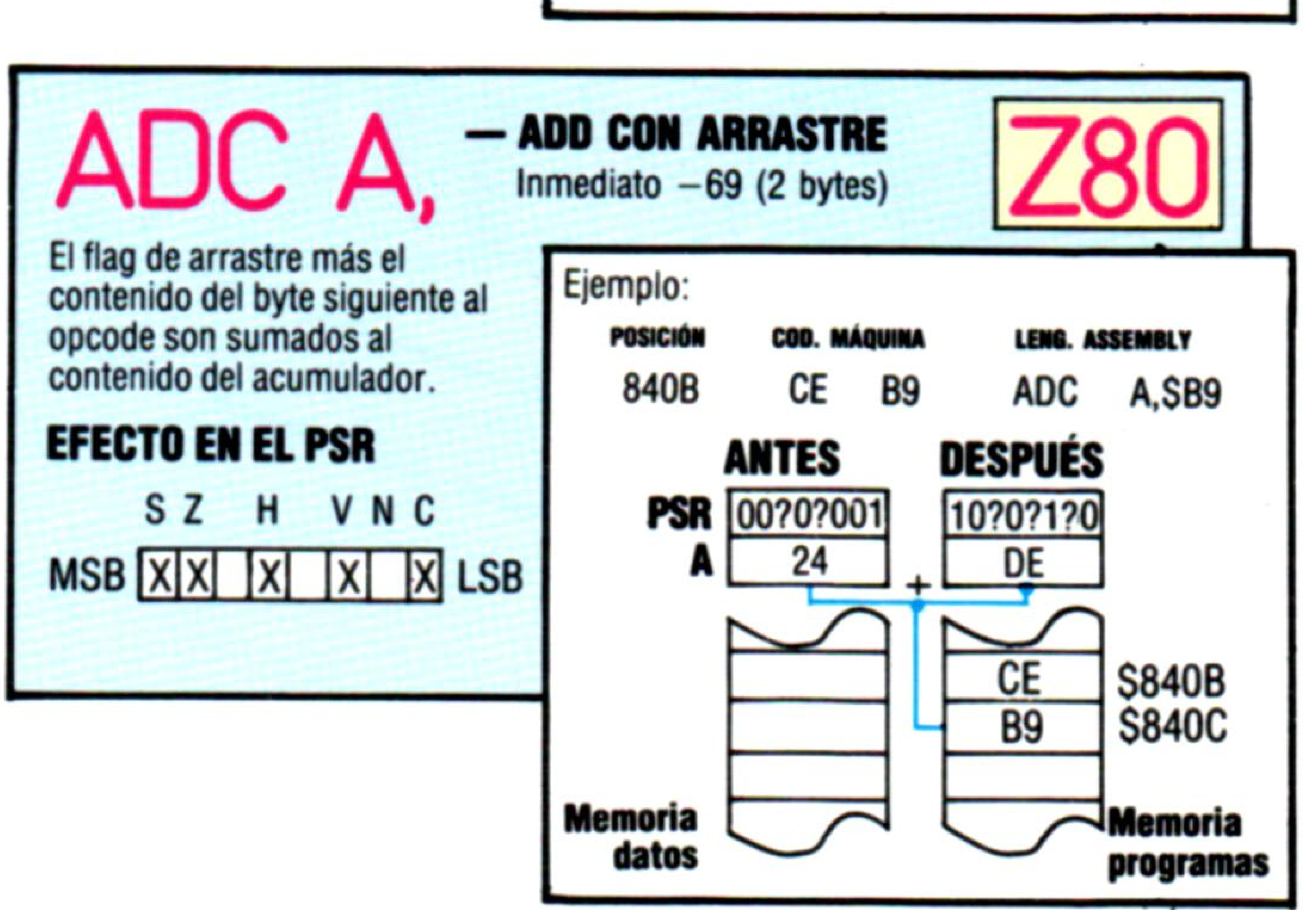

LDA
– CARGAR EL ACUMULADOR
Desde la memoria –AD (3 bytes)
6502
El contenido de la posición de memoria, cuya dirección va después del opcode, se carga en el acumulador.
EFECTO EN EL PSR
S V B D I Z C
MSB X X LSB
Ejemplo:
POSICIÓN
COD. MÁQUINA
LENG. ASSEMBLY
6F00 AD 07 B3 LDA $B307
ANTES
DESPUÉS
PSR
A
????????
??
1?????0?
D2
$B307
D2
AD
07
B3
$6F00
$6F01
$6F02
Memoria datos
Memoria programas


LD A,
– CARGAR EL ACUMULADOR
Desde la memoria –3A (3 bytes)
Z80
El contenido de la posición de memoria, cuya dirección va después del opcode, se carga en el acumulador.
EFECTO EN EL PSR
S Z H V N C
MSB LSB
SIN EFECTO
Ejemplo:
POSICIÓN
COD. MÁQUINA
LENG. ASSEMBLY
6F00 3A E9 F4 LD A,($F4E9)
ANTES
DESPUÉS
PSR
A
????????
??
????????
8C
$F4E9
8C
3A
E9
F4
$6F00
$6F01
$6F02
Memoria datos
Memoria programas


STA
–ALMACENAR ACUMULADOR
A la memoria –8D (3 bytes)
6502
El contenido del acumulador se carga en la posición de memoria cuya dirección va después del opcode.
EFECTO EN EL PSR
S V B D I Z C
MSB LSB
SIN EFECTO
Ejemplo:
POSICIÓN
COD. MÁQUINA
LENG. ASSEMBLY
E532 8D A2 65 STA $65A2
ANTES
DESPUÉS
PSR
A
????????
68
????????
68
$65A2
68
8D
A2
65
$E532
$E533
$E534
Memoria datos
Memoria programas


LD( ),A
– CARGAR EL ACUMULADOR
A la memoria –32 (3 bytes)
Z80
El contenido del acumulador se carga en la posición de memoria cuya dirección va después del opcode.
EFECTO EN EL PSR
S Z H V N C
MSB LSB
SIN EFECTO
Ejemplo:
POSICIÓN
COD. MÁQUINA
LENG. ASSEMBLY
E532 32 E9 F4 LD ($F4E9),A
ANTES
DESPUÉS
PSR
A
????????
4B
????????
4B
$F4E9
4B
32
E9
F4
$E532
$E533
$E534
Memoria datos
Memoria programas


ADC
– ADD CON ARRASTRE
Inmediato –69 (2 bytes)
6502
El flag de arrastre más el contenido del byte siguiente al opcode son sumados el contenido del acumulador.
EFECTO EN EL PSR
S V B D I Z C
MSB X X X X LSB
Ejemplo:
POSICIÓN
COD. MÁQUINA
LENG. ASSEMBLY
840B 69 A2 ADC #$A2
ANTES
DESPUÉS
PSR
A
00????01
16
11????00
B9
+
$840B
69
A2
$840C
Memoria datos
Memoria programas


ADC A,
– ADD CON ARRASTRE
Inmediato –69 (2 bytes)
Z80
El flag de arrastre más el contenido del byte siguiente al opcode son sumados al contenido del acumulador.
EFECTO EN EL PSR
S Z H V N C
MSB X X X X X LSB
Ejemplo:
POSICIÓN
COD. MÁQUINA
LENG. ASSEMBLY
840B CE B9 ADC A,$B9
ANTES
DESPUÉS
PSR
A
00?0?001
24
10?0?1?0
DE
+
CE
B9
$840B
$840C
Memoria datos
Memoria programas

Informática pop

Virgin Games ha establecido significativos lazos entre el mercado discográfico y el de software de juegos

A principios de la década de los setenta, cuando el mercado de software para ordenadores personales era una incógnita, algunos jóvenes emprendedores tuvieron en Gran Bretaña muchas posibilidades de obtener beneficios de la demanda de software en cassette. Todo el que fuera capaz de escribir divertidos programas de juegos en BASIC podía conseguir un magnetófono de alta velocidad para grabar de cassette a cassette y ofrecer su venta por correspondencia en los anuncios del diario local.

Actualmente las diversas empresas de software crean sus productos de distintos modos. Por ejemplo, Imagine Software (véase p. 559) cuenta con muchos programadores de plantilla que codifican las ideas creativas de sus jefes.

Nick Alexander, el joven director-gerente de Virgin Games, afirma: "Con frecuencia, cuanto mejor es el programador, menos ideas tiene. Para ser un buen programador hace falta ser muy lógico, muy metódico, muy diligente, y no suelen ser éstas las cualidades de un individuo creativo". Por ese motivo ofrece un servicio técnico y creativo para corregir las deficiencias de los múltiples programas que cada semana les envían jóvenes promesas. Se ayuda a los programadores capaces a desarrollar ideas y a los creativos en la codificación.

La recompensa por ser editado por Virgin podría parecer inferior a las de otras empresas. Un juego publicado por Virgin obtiene para su autor un anticipo que media entre mil y tres mil libras esterlinas, además del 7,5 % en derechos de autor sobre el precio neto. Compare estas cifras con el 25 % de derechos de autor que muchos otros editores de software afirman ofrecer. Pero Alexander sostiene que debido a que cerca del 25 % de los ingresos netos de cualquier juego son reinvertidos en su promoción, las ventas son muy superiores.

La promoción de productos es, sin lugar a dudas, una actividad que Virgin conoce al dedillo. La marca Virgin se estableció gracias a sus empresas de éxito en la industria musical y las técnicas que aprendió en ese campo Richard Branson, jefe de Virgin de treinta y un años, han sido aplicadas su ramificación en la esfera del software. Los autores de juegos son promovidos como estrellas por derecho propio: la información que aparece en cada cassette no sólo pone el nombre del autor, sino también una foto y una breve biografía. Virgin Games, que comenzó en febrero de 1983 solicitando juegos en las revistas dedicadas a los ordenadores personales, recibió alrededor de 500 ofertas iniciales. Ahora en su lista figuran 46 títulos para ocho ordenadores personales. Sus mayores ventas corresponden al Spectrum; el BBC Micro ocupa el segundo lugar, seguido por el Commodore 64.

El más reciente escritor de software contratado por Virgin es Martin Wheeler, un chico de quince años que acaba de crear dos nuevos juegos para el Spectrum, titulados *Dr Franky and the Monsters* (El doctor Franky y los monstruos) y *Sorcerers* (Brujos). Wheeler ha ensamblado los programas en lenguaje máquina y desarrollado algunos elementos gráficos impresionantes. Alexander encuentra semejanza entre el programa de los ordenadores personales de hoy y el negocio musical de hace una década, y se muestra convencido de que la informática está en vías de desplazar a la música como actividad de tiempo libre de la juventud.



Nick Alexander

Richard Branson

mi computer mi computer mi computer mi computer

Editorial Delta, S.A.





10033
9 788485 822836